私立東海大學資訊工程研究所

碩士論文

指導教授：周忠信

基於不同開發語言的軟體生產力實證探討

An Empirical Study of Software Productivity for Different Computer Languages

研究生：陳宜蓁

中華民國一〇〇年七月七日

摘要

軟體生產力的探討，對於軟體開發團隊而言頗為重要。透過生產力分析，才能夠有效獎勵團隊成員、同時改善專案進度、甚至提高軟體品質。本研究以實證方式，探討某案例公司旗下三條不同軟體產品線的生產力分析。該三條軟體產品線，除軟體流程使用的文件技術與開發用電腦語言技術不同外，其餘從軟體產品線角度而言，皆大同小異。此三條軟體產品線分別使用 Java、MS.Net、與 4GL 技術。從實證分析中發現，無論就個別應用樣式或整體模組比較，以 Java 軟體產品線的生產力最差。但若排除文件閱讀時間，僅專注於程式撰寫與單元測試，Java 軟體產品線的生產力卻是最佳。本案例研究結果指出，軟體生產力雖與採用技術有關，但更受其軟體流程影響。

關鍵字：軟體生產力、軟體產品線、軟體流程、應用樣式

Abstract

Software productivity study for a software development team is quite important, since it can be used to effectively reward team members, to help controlling project progress, and even to improve software quality. In this thesis, three different software product lines are empirical studied. Except the differences of implementation languages and documentation technologies, the three software product lines are sharing the same software life cycle management methodology. The languages for the three software product lines are Java, MS.Net, and 4GL, respectively. From our empirical study, it shows that regardless of pattern-wise or project-wise, the productivity of Java software product line is truely the worst one. However, if excluding the effort caused by the documentation technology, Java software product line demonstrates the best productivity. This study concludes that, software productivity of a software product line is not only affected by the implementation technology, but also by the software process it adopts.

Keywords: software productivity, software product line, software processes, applications pattern

表目錄

圖目錄

第一章、研究動機與目的

1.1 研究動機與目的

軟體生產力是軟體公司生存的一個重要議題。軟體生產力度量不僅是成本[9, 15, 29]與時程[4, 5, 41]估計的重要資訊來源，同時，它亦提供專案管理者有關專案狀態評量與預測之決策資訊。所以，對一個軟體公司而言，生產力是軟體開發的重要指標。高生產力意味著能夠快速且以較低成本開發軟體，因此，探討軟體生產力在軟體工程上乃至關重要。

要探討生產力，首先必須定義量測生產力的度量，在有明確的度量指標以後，才能從工程面向分析生產力，從而依據分析的結果來改善生產力[42]。近年來，雖然軟體生產力研究者眾[11, 22, 23, 29]，但仍有許多困難之處:

1. **生產力難以被正確度量。**

   傳統生產力的定義，是計算一個單位的產出量，除以每單位投入的人時[13]。而代表軟體產出量的度量單位，僅管可採用程式行數或需求功能點個數，但此兩數量仍難以全面代表產出量[26]。

2. **影響生產力因素眾多且難以評估。**

   生產力是會受許多因素所影響，其中很多是本來就難以評估的，例如:任務難度、開發團隊技能等，這些都是影響生產力的相關因素[26]。

3. **生產力與品質聯動。**

   生產力探討通常未考量品質，軟體生產力較高並不等於擁有高品質的軟體，因此，在比較生產力時，應同時將該產品品質納入分析[11]。

由於生產力指標對各產品而言，不同產品可能使用不同的生產力度量。即使使用相同度量，也不代表產品都能在同樣客觀條件下做比較。因此，為了公平比較不同產品間生產力，以下因素必須納入考慮[9，15]：

1. **開發團隊(Team):** 開發過程中，參於專案的團隊、人數、與技能水平。
2. **開發語言(Development Language):** 開發專案當中，所使用的主要電腦語言。
3. **應用領域(Application Type):** 專案開發的產品類型，主要的類型有系統軟體、應用軟體、轉鑰系統、系統整合、專業服務、處理服務、網路服等。
4. **軟體流程(Software Process):** 即軟體開發過程中，整體選用的開發程序與文件技術，例如: 瀑布式(Waterfall)、雛型式(Prototyping)、螺旋式(Spiral)、或迭代式(Iterative)等流程；文件技術，例如 E-R Modeling、DFD、或是 UML 等。
5. **CASE 工具使用(CASE Tool Used):** 開發過程中使用的任何CASE(Computer-Aided Software Engineering)開發工具，皆可能影響產出速度或品質。

本研究係以國內一家大型軟體公司的三個軟體產品，作為實證研究對象。主要目的想探討不同軟體產品的生產力比較。該軟體公司具有 28 年歷史，為台灣最大的企業應用軟體供應商。研究中所使用的數據，皆源自於該公司三個真實產品中，所收集到的數據。三個產品皆支持相同業務領域，而團隊規模類似，品質也具同樣水準。此三個產品開發皆遵循「軟體產品線」(Software Product Line, SPL)精神，具有同樣的軟體架構、相同的「應用樣式」(application patterns)、類似但不同的軟體流程(主要在於不同的分析與設計文件技術)、以及不同的開發語言：分別是 Java，MS.NET，以及 4GL (Fourth Generation Language)。根據案例公司提供數據，該公司質疑 Java 產品線之生產力較差，而本研究將進一步分析此推論是否屬實。

本研究分從兩方面著手：首先，就應用樣式方面來做比較。由於程式開發已受制於應用樣式，因此比較三條產品線在產出同樣式的功能需求下所投入的人力(effort)，可做為三條產品線的生產力比較之一。此比較目的是以個別樣式的生產力做比較。第二方向，則是以實際完成模組來進行生產力比較。透過完成一個模

組所需使用到的樣式種類與個數，各產品線所需投入的總人力可做為產品產出的生產力比較。

在經過實證探討後，發現在以應用樣式為基準的生產力比較中，無論排除或不排除非編碼因素，Java 產品線的生產力確實皆差人一等。但在以整體模組產出為基準的比較中，若排除其他非編碼因素，Java 產品線卻具有最佳的生產力。因此，本論文認為在此案例中，影響生產力最主要的因素，應該不在程式語言而是所選擇的軟體流程。

1.2 論文架構

本論文架構如下：第二章介紹與軟體生產力相關的已知研究。第三章為本案例介紹並說明所使用的生產力度量。第四章為基於樣式的生產力比較。第五章為基於整體模組的生產力比較。第六章則介紹本研究之數據局限性和研究有效性。第七章為本論文結論與未來可能研究方向。

第二章、研究背景

本研究主要在探討不同軟體產品線中的生產力比較，因此在本章中首先介紹軟體產品線的概念。同時本章也將介紹生產力的計算公式，並進一步探討現有論文對計算公式參數的定義，分析選擇不同參數的使用時機。

2.1 軟體產品線(Software Product Line)簡介

「軟體產品線」 (Software Product Line)，系指一種軟體工程技術，此技術針對特定的領域，建立一個既有的核心資產，依此開發類似或同性質高的軟體系統[18]。

軟體產品線的開發分成三步驟，分別是軟體產品線分析、設計與實作。產品線分析的目的，在於決定產品線要開發的產品種類。分析過程中，首在定義產品的領域並決定需求。而產品線設計的部份，則會針對這些需求提出實現方式，譬如使用的開發系統流程與系統架構等。產品線實作則會根據這些設計，實作相關軟體資產。

軟體產品線與一般軟體開發方式不同處在於：

1. **非客制化開發：**它不是依照各別系統的需求分別進行開發，也就是說它是基於市場主要需求為開發特性。
2. **軟體產品線不是通用的：**它是針對某個特定領域而開發，其產品針對性強。
3. **高重用性的核心資產：**軟體產品線的開發，是在一個現有軟體資產集合的基礎上，盡可能重用現有軟體資產，從而開發出一系列相似但能符合不同客戶需求的產品。

由於軟體產品線的特殊性，其基本過程主要包含兩個主要階段。第一個階段

稱為領域工程(Domain engineering)。以開發可複用核心資產為目標，目的在於滿足通用需求。第二個階段稱為應用工程(Application engineering)，通過重複利用核心資產，開發符合客戶特殊需求的最終產品。在此過程中不斷回饋領域工程，確保有效維護核心資產[40]。

如圖 2.1 所示，在軟體產品線的生產框架中，從領域工程到應用工程兩個階段當中，存在九個子流程。其中領域工程多一個產品管理的子流程，其餘八個子流程互相匹配成四組類似流程，分別是需求(requirement)、設計(design)、實作(realization)和測試(testing)在領域工程和應用工程都存在，每一匹配組的流程都緊密相連。

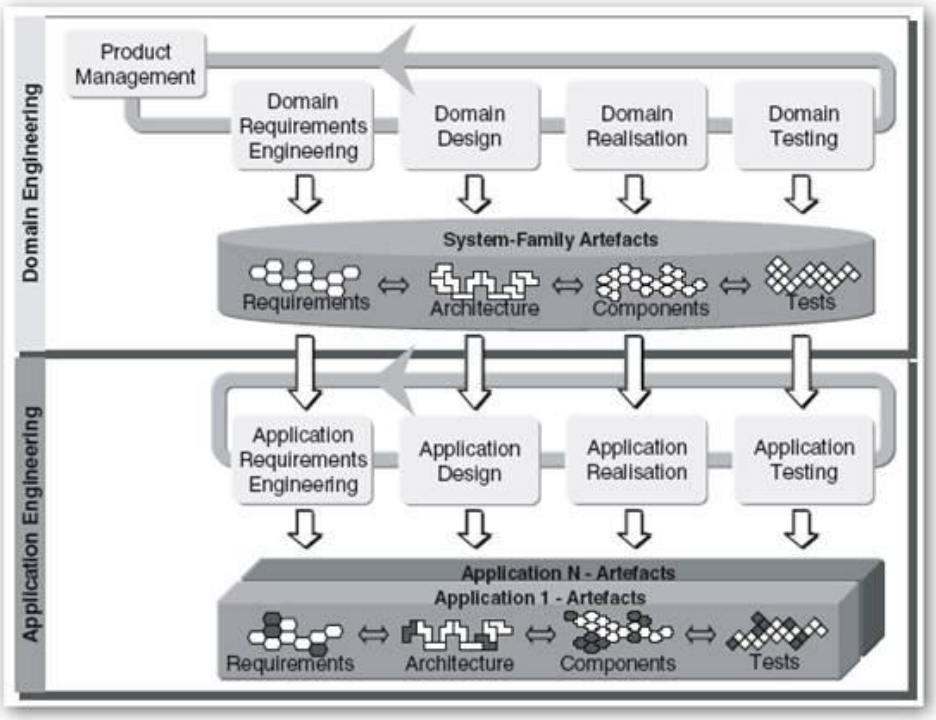

圖 2.1 軟體產品線的生產框架[40]

軟體產品線方法與傳統項目開發的主要不同，在於關注點的轉移－從單獨產品到產品線。所以，產品線的開發策略，主要依循四個基本原則，根據這些原則，將特定的專案開發轉化成特定業務領域產品。此四個基本原則分別為：

1. **可變性管理(variability management)**：每個產品都是核心資產的變體，必須系統化的管理產品的可變性。

2. **商業驅動(business-centric)**：此原則主要在決定哪些功能應該包括在產品線中，這在產品線中叫確定範圍（scoping）。

3. **架構驅動(architecture-centric)**：產品線工程依賴一個通用的參考架構（reference architecture），特定專案架構都基於參考架構進行開發。

4. **兩段生命周期(two-life-cycle approach)**：每個產品基於平台開發，產品和平台有各自的開發團隊和開發生命週期。如果條件不允許同時做這兩種工作時，自己一定要清晰自己所做的工作哪些是領域工程、哪些是應用工程。

使用軟體產品線的主要目的，在於可以充分利用共同性和管理可變性，創造和利用一個產品線中的軟體資產，以減少時間、精力、與成本的浪費。

2.2 生產力的定義

生產力對軟體開發而言，主要在衡量軟體公司資源投入(input unit)與產出(output unit)間的經濟效益[13]。傳統生產力的計算公式，為產出除以所有投入的資源。而軟體開發過程中，最大的投入資源，為整個開發時程及其投入的人力(effort)，因此軟體生產力公式可寫成如下：

$$productivity = \frac{output\ \ unit}{effort} \qquad (1)$$

在此公式中，投入資源是較不具爭議性的一個度量單位，通常以專案所耗費的總人時(亦可為人月)為其投入單位[11, 29]。但在產出單位的參數選擇上，則具有許多分歧的意見。主要分為兩種派別，以實體行數來做測量依據－大小屬性的度量(size-related measures)[29]；以及著重產品所提供之需求功能個數－功能屬性的度量(functionality-related measures)[29]。

- **大小屬性度量(size-related measures)**：大小屬性可直接對程式實體進行

度量。例如程式碼行數、物件耦合度、巢狀迴圈深度等，但多為技術層次資料。

- **功能屬性度量(functionality-related measures)**：功能屬性為軟體產品的功能點或其他與功能點相關的度量方式。無法直接對其進行度量，需要透過現實檢核間接度量。

上述兩類在研究上各有其簇擁者，本研究整理此兩類度量的相關研究及其優缺點比較，詳見表 2.1。

表 2.1 兩類屬性度量的比較

度量類型	實例	缺點	優點
大小屬性的度量(size-related measures)	1. 程式碼行數(SLOC)[3][4][5][6][7][8][9] 2. 遞送源指令(delivered source instructions; DSI)的數量[10]。 3. 模型(models)除以人力的數值[11]。 4. 類別(classes),方法(functions), 和欄位(files)的數量。	1.當專案使用不同語言開發時，無法做比較[15][20]。 2.當產出單元使用 SLOC 時，結果將受編碼風格影響[19]。 3.SLOC 無法反映出程式的複雜性與實用性[15][20]。	1. 計算簡單且具有形性: 可自動計算且易於解釋和理解[15]。
功能屬性的度量(functionality-related measures)	1. 功能點(function points)除以人力[2][11][12][13][14][15]。 2. 使用案例點(use case points)除以人力[18]。 3. 使用經驗點(use Experience Points 2.0) [39]。 4. 其他幾種不同的功能點計算方式，如 IFPUG, Mark II, NESMA, Feature Points, COSMIC[16][17]。	1.計算複雜且缺乏明確性:計算以人工方式，可能具有主觀評價方式[2][15][19][21][22]。 2.不適用於即時系統[17].(近期已有論文提出，功能點適用於即時系統的方法，e.g., COSMIC FP[25])	1. 克服了 LOC 的困難，比 LOC 更精確的做出預估[24][20]。 2. 不受語言限制，不同語言可以使用功能屬性作度量[15][29][30][31]。

大小屬性度量主要以計算程式碼行數為其產出單位，此種計算方式曾一度廣

泛被使用在生產力計算中。此度量最大優勢是計算簡便，且結果顯而易見。其適用時機，在比較兩種應用相同程式語言的專案時，大小屬性度量或可為較佳度量單位。然而，當軟體開發使用不同語言時，功能屬性度量可以不受程式碼行數或編碼風格所受限。以計算功能點當作單位產出量，雖然功能點不像程式碼行數一樣具有精確的數據。但一般而言，若經過專家所計算且認可的功能點，則是具有公信力的計算，對使用不同開發語言的專案比較，無疑是一種較公平的比較方式。

在本研究中，根據表 2.1 比較，考量三條軟體產品線使用之開發語言不同，本研究將使用功能屬性度量做為產出單位，並依此發展生產力計算公式。

第三章、案例介紹

本研究案例數據皆由台灣最大軟體公司所提供，以下簡介此軟體公司。案例公司已通過 CMMI ML3 級認證和 ISO9000 標準。公司的經營範圍涉及企業應用領域，主要領域涵蓋製造、流通、金融等行業。該公司規模約為 3000 人，遵循該公司自定義開發流程、設計、編碼、與測試等規範。本次討論系以其旗下的三條軟體產品線為目標。

此三條軟體產品線遵循軟體產品線概念，使用程式語言分別是：PL-J 產品線使用 Java、 PL-N 產品線使用 Microsoft.NET C#、以及 PL-G 產品線使用 4GL (Fourth Generation Language)。此三條軟體產品線同具以下幾種特性:

- 以開發企業應用軟體為主，三條軟體產品線具有共通的商業領域。
- 相同的領域工程與應用工程，但開發技術不同。
- 三條軟體產品線遵循同一參考架構，但使用不同發展技術。

該案例公司之所以建立三條軟體產品線，是為滿足客戶的技術偏好。因此，它們具有類似的軟體架構。以下根據第一章所提出影響生產力的五點因素為基礎，分別對於三條軟體產品線作分析：

表 3.1 三條軟體產品線的開發背景

軟體產品線	PL-J	PL-N	PL-G
應用領域	企業資源規劃	企業資源規劃	企業資源規劃
人數	27	33	40
開發語言	Java	C#	4GL
IDE	Eclipse	Visual Studio	Genero
軟體流程	相同	相同	相同
文件技術	UML	自訂	自訂
CASE 工具	未使用	未使用	未使用

三條軟體產品線當中，PL-G 建立已有十年以上的歷史，故在三條軟體產品線中較為成熟。

3.1 背景分析

從前面三條軟體產品線簡述中可以得知，影響生產力因素的幾點中，三條軟體產品線除了軟體流程使用的文件技術與開發語言不同外，三者並無太大差異。然而在比較三者生產力之前，必須更明確確認三者間比較的公平性。

- **相同的軟體架構**

  三條產品線的軟體架構是相似的，三者皆以同樣的應用樣式為開發基礎。所謂的應用樣式，是為適用於企業應用領域所發展的一門樣式類型[21]，主要用來解決開發企業應用軟體中所發生的常見問題。根據該軟體公司過去開發經驗，共有十一種應用樣式。基於商業機密，不在此詳述此十一種樣式內容。但案例中之三條產品線，都遵循這些樣式來開發產品。為方便陳述，本論文將此十一種樣式從一到十一予以編號，分別以 P_n 表示第 n 個應用樣式。

- **相同等級的品質**

  由於生產力無法表示出軟體品質的優劣，因此在比較生產力時，軟體品質是必須納入考量的一環。同時，就不同團隊而言，開發人員的技能等級，也是必須考量的因素。因此，就品質方面，本案例分從以下兩個面向說明：

  - 質量方面：本案例公司通過 CMMI ML3 級的認證，在軟體品質要求上具有相同標準，因此該三條軟體產品線具有相似的品質等級。
  - 人員方面：本案例公司三條產品線工程師，都有 3-7 年的編碼經驗。所有團隊成員也達同等技能水平。當工程師被錄用時，都必須接

受同等量的訓練。

- **不同的文件技術**

三條軟體產品線最主要的差異處，除了使用不同的技術語言開發外，在流程上還使用不同的文件技術。其中 PL-J 使用 UML 來描述系統分析與設計文件。比起其他兩線基於自身經驗所制訂的文件，遠較為複雜。其原因可能在於對 UML 的不熟悉，或是物件導向技術本身即不適用於軟體產品線。

3.2 工作項目分類

該公司根據以往歷史數據，統計程式設計師在主要三種工作項目的耗費工時比。此三個工作項目分別為：文件閱讀(document reading)，編碼(coding)和單元測試(unit testing)。表 3.2 為該公司三條軟體產品線三種工作項目的工時耗費比率。

表 3.2 三種產品線的工時耗費比率

類型	PL-J	PL-N	PL-G
文件閱讀 (Document Reading)	31.40%	21.30%	14.20%
編碼 (Coding)	39.60%	53.10%	58.80%
單元測試 (Unit Testing)	29.00%	25.60%	27.00%

3.3 生產力定義

在第二章中已介紹軟體生產力的公式為產出單位除以人力，在本節中將根據本案例參數修改生產力公式。

首先來看生產力的分子部份。在本研究中，三條產品線的產品類型一致。同時，該公司根據應用功能歸類出十一個應用樣式。每個樣式再根據欄位(field)個數多寡，可以預估出功能點數。此種作法能夠非常準確預估每次應用需求的功能大小。而更重要的是，由於三條產品線皆採用同一技術與方法，因此在生產力比較上，可以確保分子的公平性。因此，在三條軟體產品線的產出部分可以做出以下定義：產出量等於函式 *f*(應用樣式, 欄位個數)。若以 *m* 代表欄位個數，*type* 代表應用樣式，則產出量可以寫成如下：

$$output\ unit = f(type, m) \qquad \textbf{(2)}$$

由於三條軟體產品線的產出量預估方法一樣，因此我們可鎖定三條軟體產品線在相同需求時，生產力公式中的分子數值是相同的，但三條軟體產品線要滿足同樣需求的人力數卻各不一樣。因此在比較三條軟體產品線的生產力時，其分子皆簡化為 1。若以 *i* 代表 PL-J、PL-N、與 PL-G 三條軟體產品線，*effort(i)*則代表 *i* 產品線針對該次需求所投入的人力。生產力公式可簡化如下:

$$productivity(i) = \frac{f(type, m)}{effort(i)} = \frac{1}{effort(i)} \qquad (3)$$

在本研究中，為避免因蒐集資料當時，不同團隊的技術水平不一，進而造成生產力比較失真。因此在 *effort(i)* 計算上，乃採用人力預估模型。人力預估主要是在預測開發某個應用需求時，所需要耗費的人力工時數 [10]。由於該公司的三條產品線都已成熟多年，同時也遵循制式應用樣式來開發產品。因此該案例公司乃根據過去開發歷史記錄，發展自定義估算模型。此模型經三條產品線專家不停修正，因此，預估投入之人力數值乃具有正確性。事實上，預估值與實際投入人力數極為接近。

為了精確且公平的比較三條軟體產品線的生產力，尚有一項議題必須納入考量。其中，程式設計師在投入人力時，耗費在三類工作項目中的工時比例不盡相

同。而文件閱讀的時間，應該與不同電腦語言所需投入的人力無直接關係，因此為了更公平比較此三條產品線的生產力，在人力投入上本研究採用兩種計算方法：

1. 整體人力(whole effort)：不排除文件閱讀的所有投入時間。
2. 編碼與測試人力(coding and testing effort)：排除文件閱讀時間，只包括編碼和單元測試時間。

生產力在上數計算方式下，又可分成以下兩種生產力公式：

$$p_W = \frac{1}{effort_W(i)} \qquad (4)$$

$$p_{CT} = \frac{1}{effort_{CT}(i)} \qquad (5)$$

數學式(4)中的 $effort_w(i)$代表 i 產品線針對該次需求所投入的整體人力，P_W 則代表以整體人力為分母所計算出的生產力比值；數學式(5) 中的 $effort_{CT}(i)$代表 i 產品線投入的編碼與測試人力，P_{CT} 則代表以編碼與測試人力為分母的生產力比值。在本研究中，將同時採用這兩種生產力公式，做為比較三條軟體產品線的工具。

第四章、基於應用樣式的生產力比較

由於案例公司的三條軟體產品線，採用相同的十一種應用樣式。因此在本章中，我們先從應用樣式角度分析三條軟體產品線的生產力。

首先，表 4.1 是由該案例公司從歷史資料中，依每個應用樣式的平均需求欄位個數，分別套用人力預估模型所計算出的平均投入人力。

表 4.1 三條軟體產品線的十一種應用樣式平均投入人力

	平均人力 (單位:人時)		
應用樣式	**PL-J**	**PL-N**	**PL-G**
$\mathbf{P_1}$	14.0	18.3	11.7
$\mathbf{P_2}$	31.0	32.5	39.5
$\mathbf{P_3}$	74.0	61.0	55.0
$\mathbf{P_4}$	47.5	31.0	24.0
$\mathbf{P_5}$	53.1	38.8	27.5
$\mathbf{P_6}$	52.0	30.5	31.0
$\mathbf{P_7}$	23.0	13.0	24.0
$\mathbf{P_8}$	19.0	10.2	14.0
$\mathbf{P_9}$	58.0	26.0	30.1
$\mathbf{P_{10}}$	18.0	8.0	8.0
$\mathbf{P_{11}}$	27.3	23.0	29.0

根據第三章中的數學式(3)，投入人力和生產力成倒數關係。因此當投入人力數值越高，其生產力相對越差。

4.1 應用樣式的整體人力比較

根據表 4.1，使用數學式(4)來比較三條軟體產品線在十一種應用樣式當中的生產力，由於數學式(4)計算的是整體人力，所以直接取用表 4.1 的數值進行分析，可得出圖 4.1 的生產力比較折線圖。

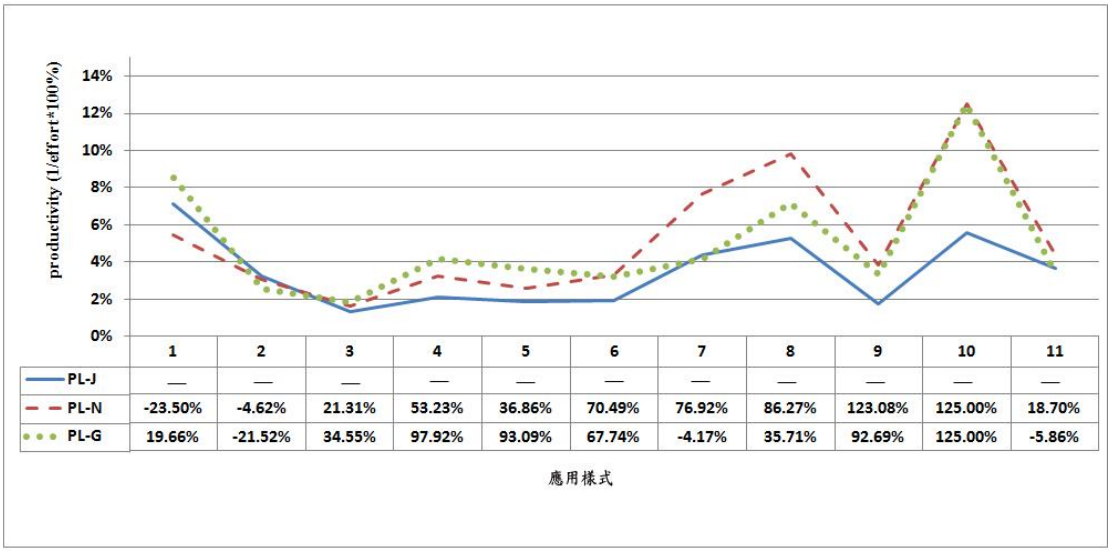

圖 4.1 三條產品線在十一種應用樣式中的整體生產力

從圖 4.1 中發現，三條軟體產品線的生產力，在十一個應用樣式中各有優劣，並無絕對勝出者。若以 PL-J 為基本線，PL-N 除在應用樣式 P_1 與 P_2 中，生產力比 PL-J 要低 23.5%與 4.62%外，PL-N 在應用樣式 P_3~ P_{11} 的生產力皆比 PL-J 高。在 PL-J 與 PL-G 的比較中，同樣以 PL-J 為基本線，PL-G 亦只有在應用樣式 P_2、P_7、P_{11} 當中，生產力較 PL-J 分別低 21.52%、4.17%、5.86%外，其他應用樣式之生產力皆比 PL-J 高。為了進一步釐清三條軟體產品線生產力的優劣，下一節將僅比較編碼人力上的生產力。

4.2 應用樣式的編碼與測試人力比較

不同產品線受應用需求影響，文件的複雜性也各有不同。閱讀文件確實可能影響程式開發時間，或是因誤解需求而造成較多單元錯誤。因此，在本節中為排除文件閱讀的不一致性，將只依據編碼與單元測試的投入人力做為比較基礎。表 4.2 為三條產品線在排除文件閱讀後的所有投入人力。

表 4.2 三條軟體產品線的十一種應用樣式編碼人力

	預估人力(單位:人時)		
應用樣式	**PL-J**	**PL-N**	**PL-G**
P_1	9.6	14.4	10.0
P_2	21.3	25.6	33.9
P_3	50.8	48.0	47.2
P_4	32.9	24.4	20.6
P_5	36.4	30.5	23.6
P_6	35.7	24.0	26.6
P_7	15.8	10.2	20.6
P_8	13.0	8.0	12.0
P_9	39.9	20.5	25.8
P_{10}	12.4	6.3	6.9
P_{11}	18.7	18.1	24.9

圖 4.2 為根據表 4.2 與數學式(5)，比較三條軟體產品線在十一種應用樣式當中的生產力，結果使用折線圖方式呈現。

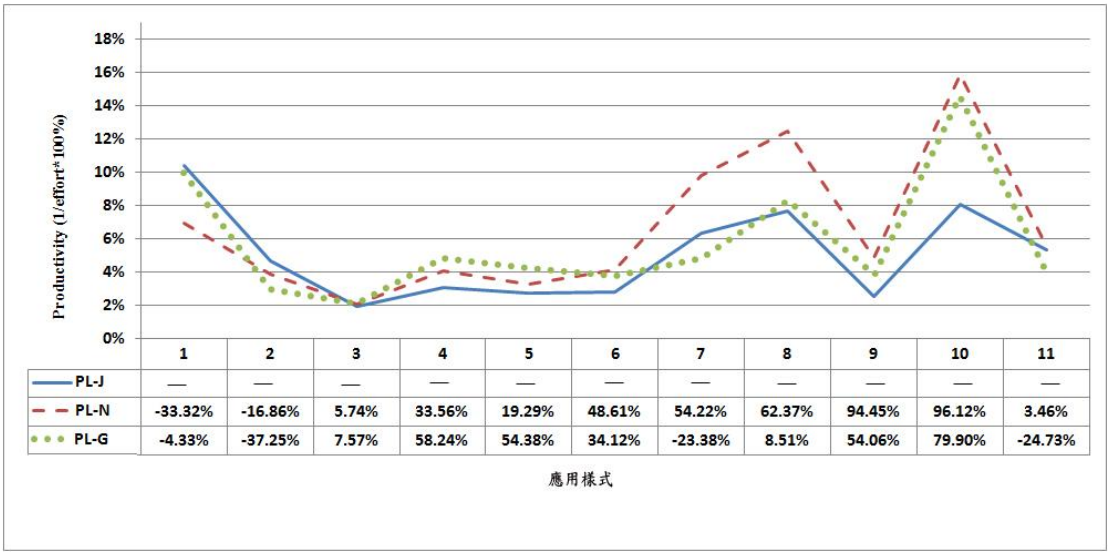

圖 4.2 三條產品線在十一種應用樣式中的編碼人力

此三條軟體產品線在僅考量編碼與測試人力下，每個應用樣式的生產力表現依舊各有優劣，難以評斷何者生產力具有絕對優勢。只能評斷若僅考量編碼與測

試人力的生產力時，三條軟體產品線的生產力差異幅度相對變小。若分開來看，以 PL-J 為基本線，PL-N 在應用樣式 P_1 與 P_2 中，生產力比 PL-J 要低 33.32%與 16.86%。在 PL-J 與 PL-G 比較中，同樣以 PL-J 為基本線，PL-G 亦只有在應用樣式 P_1、P_2、P_7、與 P_{11} 中，生產力較 PL-J 分別低 4.33%、37.25%、23.38%、與 24.73%。

4.3 產品線的生產力比較

從上兩節來看，單從個別應用樣式比較三條軟體產品線的生產力，十一個應用樣式之中，並無一條產品線具絕對優勢。本節試圖將十一種應用樣式視為一個整體，進行三條軟體產品線的生產力比較。

首先，將產品線的生產力視為一個向量，該向量由十一個應用樣式代表的個別維度所組成。而每個產品線對應於應用樣式的投入人力數值，則視為生產力向量在該維度的分量。生產力向量的絕對長度，或是該點與原點的距離，則視為該產品線基於應用樣式的生產力。若生產力向量的絕對長度越大，代表其在應用樣式整體面的生產力優勢也越高。以 P_i 表示第 i 類應用樣式，$effort(P_i)$代表 P_i 針對該次需求所投入的人力，生產力優勢計算方式如下：

$$\sqrt{\sum_{i=1}^{11}\left(\frac{1}{effort(p_i)}\right)^2} \qquad (6)$$

圖 4.3 為根據數學式(6)與表 4.1 數據，計算出三條產品線在應用樣式整體面的生產力優勢比較。從圖中發現，PL-N 與 PL-G 的生產力優勢類似，而 PL-J 的生產力優勢則相對較差。

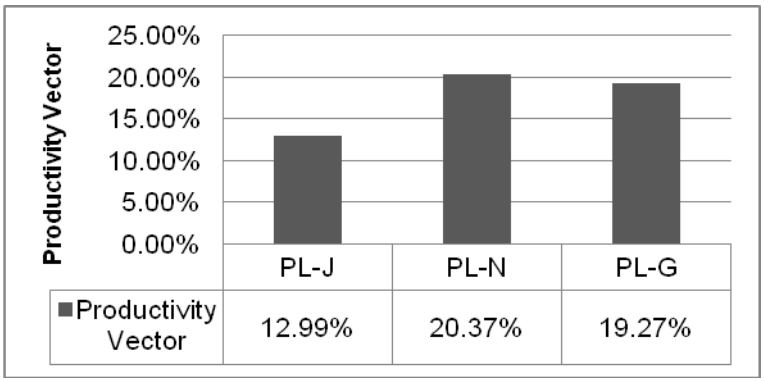

圖 4.3 三條軟體產品線整體生產力優勢分析

圖 4.4 為根據數學式(6)與表 4.2 數據，計算出三條產品線在排除文件閱讀下的應用樣式整體面生產力優勢比較。雖然 PL-J 的生產力優勢仍然略遜於其他兩個產品線，但三者已無太大差異。

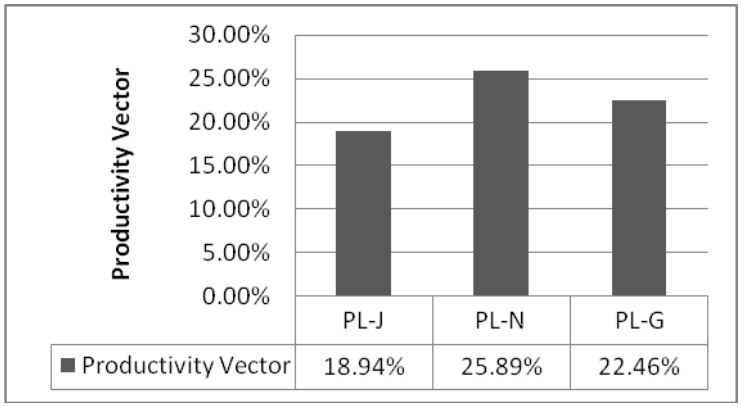

圖 4.4 三條軟體產品線排除文件閱讀之生產力優勢分析

根據本章的比較與討論，無論從個別應用樣式或是整體應用樣式，以及排除或不排除文件閱讀來看，三條軟體產品線中，PL-J 產品線確實皆呈現較差的生產力。換言之，該案例公司懷疑 Java 技術造成生產力低落，顯然信而有徵。為更確定此懷疑的正確性，下一章將從各產品線完成一個整體模組的角度，來探討三個產品線的生產力問題。

第五章、基於專案的生產力比較

前一章中，三條軟體產品線在應用樣式分析上，生產力以 PL-N 最優，PL-G 次之，PL-J 最差。然而在現實生產環境中，一個完整模組會使用到多個不同應用樣式，而每個應用樣式的欄位個數亦不盡相同。因此，要更清楚比較三個產品線的生產力，應該考慮現實專案中產出整體模組的投入人力。

根據該案例公司所提供的歷史數據，表 5.1 所列為一個標準專案中，十一個應用樣式的平均佔有比例。

表 5.1 標準專案中的十一種應用樣式平均佔有比例

應用樣式	標準專案中佔據比例
P_1	37%
P_2	30%
P_3	3%
P_4	2%
P_5	10%
P_6	2%
P_7	2%
P_8	4%
P_9	2%
P_{10}	3%
P_{11}	5%

為有效比較三個產品線差異，本文定義標準專案的平均生產力如以下數學式(7)所示：

$$\frac{1}{\sum_{i=1}^{11} effort(p_i) \times R_i} \qquad (7)$$

其中，R_i 為每個應用樣式 P_i 在標準專案當中所佔的比例，$effort(P_i)$定義同數學式 6。分母乘積的總合為一個標準專案的平均投入人力，而其倒數則用來代表

標準專案的平均生產力。

5.1 整體人力比較

根據數學式(7)、表 4.1 與表 5.1，圖 5.1 為三條產品線的標準專案之平均生產力比較。同樣地，PL-J 產品線仍然最差，PL-G 則是三者中生產力最佳的產品線。換言之，從專案角度來看，此結論頗符合該案例公司的直覺感受。

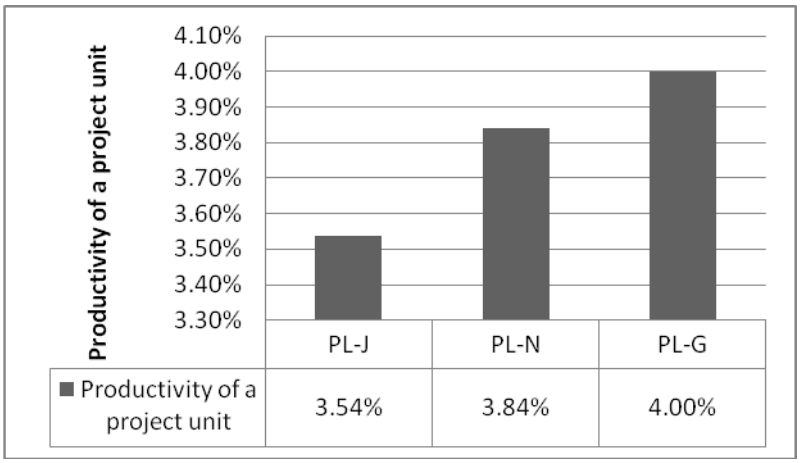

圖 5.1 三條軟體產品線之標準專案平均生產力比較

5.2 編碼與測試人力比較

同樣地，圖 5.2 是根據數學式(7)、表 4.2 與表 5.1 所發展的三條產品線之標準專案平均生產力比較。跟前面所有分析探討非常不同的地方是，PL-J 產品線的平均生產力已高於 PL-N 及 PL-G 的平均生產力。在排除文件閱讀的情況下，PL-J 的生產力由原本的 3.54%提升到 5.15%，成為三者之冠。PL-N 和 PL-G 的生產力提升幅度較小，分別由 3.84%提升到 4.88%，以及由 4%提升到 4.66%。換言之，從專案角度來看，若排除文件閱讀時間，PL-J 產品線一點都不輸給其他兩個產品線，此結論與該案例公司的直覺感受正好相反。

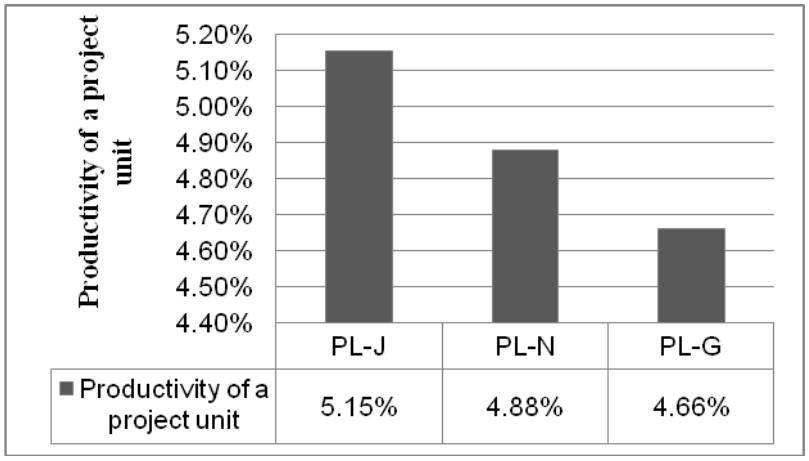

圖 5.2 三條軟體產品線排除文件閱讀之標準專案平均生產力比較

第六章、研究限制與驗證

在本研究中，有三點可能影響研究結果的可信度，需要加以驗證。

第一點，案例軟體公司提供的應用樣式預估人力表 4.1 之正確性。由於其所提供的預估人力，是以該公司一般常見產品的統計數值為樣本，擷取樣本時的公平性是否合理有待商榷。若出現爭議時，將影響本研究有關生產力比較的正確性。此數據已獲得案例軟體公司確認無誤。

第二點，案例軟體公司提供的應用樣式在專案中佔有比例，表 5.1 內容之正確性可能需要被驗證。此部分透過專家取得，可能受個人經驗和主觀判斷影響，造成數值偏差。在研究進行時，此部分數據已通過三條產品線管理階層認可，認定為可靠數據，所以應具實際意義。

第三點，案例軟體公司所提供的程式設計三類工作細項所佔之耗費工時比例，亦即表 3.1，準確度可能存在偏差。但此部分資料是經由該公司之歷史數據，內部統計分析出的結果，同時亦經由三條軟體產品線之管理階層認可，所以應具實際意義。

第七章、結論

本研究以實證方式，探討某案例公司旗下三條不同軟體產品線的生產力分析。該三條軟體產品線，除軟體流程使用的文件技術與開發用電腦語言技術不同外，其餘從軟體產品線角度而言，皆大同小異。無論從個別應用樣式或是整體應用樣式，三條軟體產品線中，以 Java 技術為主的 PL-J 軟體產品線確實呈現較差的生產力。故案例公司對 Java 生產力的懷疑，在此處可以得到驗證。不過，若排除文件閱讀，三條軟體產品線中，雖然仍是 PL-J 軟體產品線生產力較差，但與其他兩個軟體產品線相較，差異並不明顯。若從專案角度來看，在排除文件閱讀而僅專注於程式撰寫與單元測試下， 以 Java 技術為主的 PL-J 軟體產品線之生產力反而最佳。根據上述分析結果，我們認為各軟體產品線生產力雖與使用技術有關，但顯然也受軟體流程影響。

第八章、參考文獻

[1]. A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," *IEEE Transactions on Software Engineering*, vol.9, pp. 639-648, 1983.

[2]. M. Arnold and P. Pedross, "Software size measurement and productivity rating in a large-scale software development department," *20th International Conference on Software Engineering*, Kyoto, Japan, 1998.

[3]. C. A. Behrens, "Measuring the productivity of computer systems development activities with function points," *IEEE Transactions on Software Engineering*, vol.9, pp.648-652, 1983.

[4]. J. D. Blackburn and G. D. Scudder, "Time-based software development," *Integrated Manufacturing Systems*, vol. 7, pp.60-66, 1996.

[5]. J. D. Blackburn, G. D. Scudder and L. N. Van Wassenhove, "Improving Speed and Productivity of Software Development: A Global Survey of Software Developers," *IEEE Transactions on Software Engineering*, vol.22, pp.875-886, 1996.

[6]. L. C. Briand, and Wieczorek, I. *Resource estimation in software engineering.* In: J. J. Marcinak, Editor, Encyclopedia of Software Engineering, New York: John Wiley & Sons, 2002.

[7]. F. Bootsma, "How to obtain accurate estimates in a real-time environment using full function points," Proceedings of the *3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, Richardson, TX, USA, 2000.

[8]. D. N. Card, F. E. McGarry , and G. T. Page, "Evaluating software engineering technologies," *IEEE Transactions on Software Engineering*, vol.13, pp.845-851, 1987

[9]. C. Comstock, Z. Jiang, P. Naudé, "Strategic Software Development: Productivity Comparisons of General Development Programs," *International Journal of Computer and Information Science and Engineering*, vol.23, pp.357-362, 2007

[10]. S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Menlo Park, California: Benjamin/Cummings Publishing Company, 1986.

[11]. N. Fenton, and S. L. Pfleeger, *Software metrics: a rigorous and practical approach*, Boston, MA, USA: PWS, 1997.

[12]. T. Hall, N. Fenton, "Implementing effective software metrics programs," *IEEE Software*, vol.14, pp.55-64, 1997.

[13]. IEEE standard for software productivity metrics, in: *IEEE Std*, 1045-1992, 1993.

[14]. R. Jeffery, M. Ruhe, and I. Wieczorek, "A Comparative Study of Two Software Development Cost Modeling Techniques Using Multi-organizational and Company- Specific Data," *Information and Software Technology*, vol.42, pp.1009-1016, 2000.

[15]. Z. Jiang, P. Naudé, and C. Comstock, "An Investigation on the Variation of Software Development Productivity," *International Journal of Computer and Information Science and Engineering*, vol.1, pp.72-81, 2007

[16]. C. Jones, *Applied Software Measurement: Assuring Productivity and Quality.* New York: McGraw-Hill, 1991

[17]. C. F. Kemerer, and B. S. Porter, "Improving the reliability of function point measurement: an empirical study," *IEEE Transactions on Software Engineering*, vol.18, pp.1011-1024, 1992

[18]. C. W. Krueger, "New methods in software product line practice," *Commun. ACM*, vol.49, pp.37-40, 2006.

[19]. G. C. Low and D. R. Jeffery, "Function points in the estimation and evaluation of the software process," *IEEE Transactions on Software Engineering*, vol.16, pp.64–71, 1990.

[20]. A. MacCormack, C. F. Kemerer, M. Cusumano, and B. Crandall, "Trade-offs between productivity and quality in selecting software development practices," *IEEE Software*, vol.20, pp.28-34, 2003.

[21]. H. Mathkour and A. Touir, "The design and implementation of an e-Business Pattern Language," *Asia-Pacific Conference on Communications APCC*, Malaysia, 2003.

[22]. K. D. Maxwell, "Collecting data for comparability: benchmarking software development productivity," *IEEE Software*, vol.18, pp.22-26, 2001.

[23]. K. D. Maxwell, L. V. Wassenhove, and S. Dutta, "Software development productivity of European space, military and industrial applications," *IEEE Transactions on Software Engineering*, vol.22, pp.706-718, 1996.

[24]. S. Morasca and G. Russo, "An empirical study of software productivity," Proceedings of the *25th International Computer Software and Applications Conference on Invigorating Software Development*, Chicago, IL, USA, 2001.

[25]. H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Systems with Applications*, vol.35, pp.929–937, 2008.

[26]. R. Premraj, M. Shepperd, B. Kitchenham and P. Forselius, "An Empirical Analysis of Software Productivity over Time," In Proceedings of *11th IEEE International Software Metrics Symposium (METRICS'05)*, Como, Italy, 2005.

[27]. R. W. Selby, *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research.* Boston, Mass: Addison-Wesley, 2007.

[28]. H. M. Sneed, "Measuring the performance of a software maintenance department." *1st Euromicro Conference on Software Maintenance and Reengineering*, Berlin, Germany, 1997.

[29]. I. Sommerville, *Software engineering*, Mass, Boston: Addison-Wesley, 2004.

[30]. C. Stevenson, *Software engineering productivity: a practical guide.* London: Chapman & Hall, 1995.

[31]. G. H. Subramanian and G. E. Zarnich, "An examination of some software development effort and productivity determinants in ICASE tool projects," *Journal of Management Information Systems*, vol.12, pp.143-160, 1996.

[32]. C. R. Symons, "Function point analysis: difficulties and improvements." *IEEE Transactions on Software Engineering*, vol.14, pp.2-11, 1988.

[33]. C. R. Symons, *Software sizing and estimating: Mk II FPA (function point analysis)*, New York: Wiley, 1991.

[34]. P. Tomaszewski and L. Lundberg, "Software Development Productivity on A New Platform: An Industrial Case Study," *Information and Software Technology*, vol.47, pp.257–269, 2005.

[35]. P. Tomaszewski and L. Lundberg, " The increase of productivity over time – an industrial case study," *Information and Software Technology*, vol.48, pp.915–927, 2006.

[36]. P. Tomaszewski, L. Lundberg, J. Håkansson and D. Häggander, "Evaluating Real-time Credit-control Server Architectures Implemented on a Standard Platform." Proceedings of *IADIS International Conference on Applied Computing*, vol. 2, pp.345-352. Algarve, Portugal, 2005.

[37]. P. Tomaszewski, L. Lundberg, J. Håkansson and D. Häggander, "A Cost-efficient Server Architecture for Real-time Credit-control," Proceedings of the *10th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pp.166-175, Shanghai, China, 2005.

[38]. P. Tomaszewski, and L.-O. Damm, "Comparing the Fault-Proneness of New and Modified Code – An Industrial Case Study," Proceedings of the *5th ACM-IEEE International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, 2006.

[39]. P. Tomaszewski, P. Berander and L.-O. Damm, " From Traditional to Streamline Development - Opportunities and Challenges," *Software Process: Improvement and Practice - Special Issue on Systems*, vol.13, pp.195-212, 2008.

[40]. F. Van der Linden, K. Schmid, E. Rommes, *Software Product Lines in Action – The Best Industrial Practice in Product Line Engineering*. Berlin: Springer Verlag, 2007.

[41]. M. Van Genuchten, "Why Is Software Late? An Empirical Study of Reasons for Delay in Software Development," *IEEE Transactions on Software Engineering*, vol.17, pp.582-591, 1991.

[42]. W. D. Yu, D. P. Smith and S. T. Huang, "Software productivity measurements," Proceedings of the *15th Annual International Computer Software and Applications Conference COMPSAC '91*, Tokyo, Japan, 1991.