專案管理與 CMMI 中執行力、執行度、和軟體品質的探討

Ability to Perform, Degree of Commitment, and Software Quality in Project Management and CMMI

陳仲儼
長庚大學資訊管理研究所
桃園縣龜山鄉文化一路 259 號
Email: cychen@mail.cgu.edu.tw

Chung-Yang Chen
Dept. of Information Management,
Chang Gung University
Email: cychen@mail.cgu.edu.tw

摘要

國內軟體業者在專案開發上常常延宕交件或超出預算，因此，專案後期與客戶的協商成了專案管理中主要工作之一。然而軟體企業在邁向國際化時，此已不再是權宜之計。本文針對此一現象，在專案管理上試圖論述解決之道。作者以為管理軟體專案的開發有三大訴求:執行度、執行力、及軟體品質，且為能力成熟度整合模式(CMMI)第二層級 Managed 之實踐。專案之執行度端賴有效的專案人員組織管理，尤其面對大型專案，或多個專案同時進行。專案之執行力端賴專案資源制度化的管理，其內容包括 CMMI 層級二之流程領域。而在全面品質管理觀念中，專案執行度及執行力則影響軟體品質。本研究試圖從抽象化物件導向專案組織架構、專案開發模式、及 CMMI Level 2 的專案管理三方面來探討軟體組織如何掌握專案執行度、執行力，進而改善軟體品質。

關鍵詞: 軟體品質、專案管理、能力成熟度整合模式

Abstract

Software industry usually has the following problem: project overdue or budget overspent; therefore, how to pacify customers has been critical in project management. Yet it is no more applicable as Taiwan software organizations become internationally competitive. To eliminate the above plausible issue, this paper suggests three aspects for managing software development: degree of commitment, ability to perform, and software quality, which are also fulfillments for Capability Maturity Model Integration (CMMI) Level 2. A solution is discussed by looking into abstract project organization, project development process, and project management in CMMI Level 2 in order to help software organizations manage degree of commitment, ability to perform, and software quality during software development.

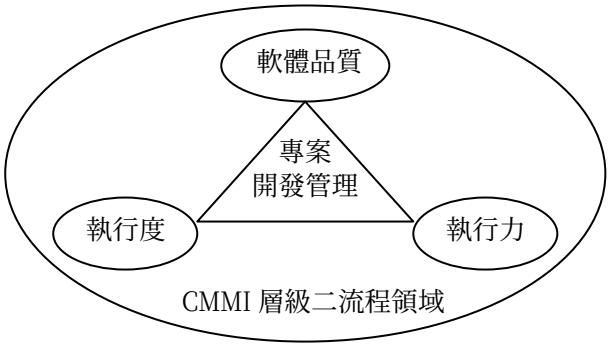
Keywords: Project Management, Software Quality, CMMI

一、專案開發管理與 CMMI

軟體工業近年來已成為台灣另一個欲爭取演出機會的國際舞台。不像傳統產業需生產機器設備般，軟體開發也許僅需要電腦和更重要的，大腦，即能從事軟體產品開發 [21]。也因為程式語言的統一，使得即使是小型的軟體專案，也可由懂得開發程式語言的關鍵人員在任何時間、任何地點完成，並且透過網路，立即將產品交給客戶。這也使得客戶能突破地理與時間限制，尋求既便宜又有效率的程式開發供應商。例如，美國最大的軟體代工商大部分集中在印度即是一例。這些軟體開發的特色也更說明了軟體工業國際化的現象與趨勢。

但也由於軟體工業非常依賴人，專案開發過程的許多問題便因 [5][7]，如需求溝通不良，關鍵人員易動，人員程式能力不足，或工作過於繁重等等。這些問題造成了人員執行度(degree of commitment)無法掌握，如延遲完成時間 [18]，及軟體品質低落(如程式上的錯誤(faults/errors)，或客戶在使用上的不滿意(problem of usability)，例如軟體雖沒有錯誤但卻沒將規格完全實作(problem of completeness)，或界面不符要求(problem of validation)。而這些因人而產生的問題正是軟體專案管理上最主要的問題與挑戰之一。

由於一件軟體複雜的組態，及層層委外、代工、發包的企業合作模式之緣故，如何讓整個軟體供應鏈上每一環節都能符合品質要求成為企業與企業間的一項中重要課題，這也把全面品質管理提升至以產品為導向的企業間組態管理。為了因應此一難題，制度化的標準模式如 CMM、CMMI、ISO、MIL-STD 498、Trillium 等，及 CMM SCAMPI、Bootstrap, ISO 15507 SPICE 等評鑑制度出爐 [6][8][14][22][23]，期能在軟體開發供應鏈上作好預知性(Proactive)品管以取代過去的因應性的措施(Responsive，即產品接手後才檢驗品質優劣，再依據協議內容作懲處)。這也說明了由於美國將 CMMI 系列產品視為標準，印度軟體業者因而普遍將 CMM 或 CMMI 制度化並作層級認證的現象[25]。



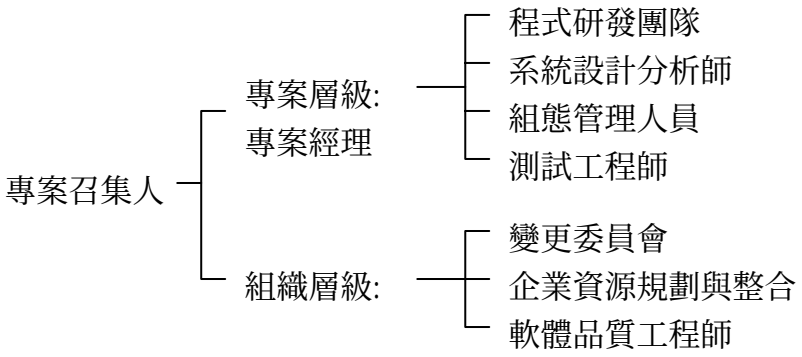
圖一: 管理專案開發的三個訴求(本研究製)

一個小規模的軟體組織不見得就是在軟體品質及專案管理上有先天的弱點。相對的，要成為一個在專案管理上小而美的軟體組織並消弭上述的現象，軟體組織需要有

一套制度化的專案管理模式。昂貴的軟體工具並非專案管理唯一方式。反而，制度化的建立能帶與組織體質上的根本改善。作者以其日前接受 SEI 的 CMMI 教育訓練認證內容中，加上目前協助軟體公司落實 CMMI 層級二的手法及研究心得，以為管理軟體專案的開發有三大訴求:執行度、執行力、及軟體品質，且為能力成熟度整合模式(Capability Maturity Model Integration，簡稱為 CMMI)第二層級 Managed 之實踐。而 CMMI 第二層級指組之織在專案開發的過程中滿足了七個流程領域(process area)抽象化規範的要求：需求管理、專案控管、組態管理、度量分析、專案規劃、產品流程品質保證、及供應商協議管理 [3][22]。專案之執行度端賴有效的專案人員組織管理，尤其面對大型專案，或多個專案同時進行時。專案之執行力端賴專案資源制度化的管理。制度化的軟體專案開可從三個主題著手：專案組織架構抽象化，開發模式結構及遞迴化，並且搭配 CMMI 層級二之落實來確保組織在規劃專案後對於執行度、執行力、及軟體品質之掌握(如[圖一])，以進一步作好預知性品管。

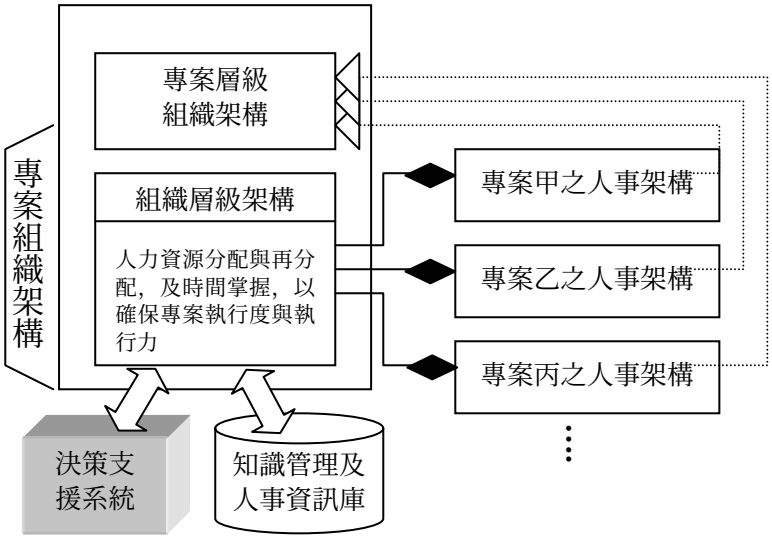
二、專案組織架構

針對每個專案計劃，軟體公司常成立一個專案團隊，並有窗口負責與客戶維持全時並且動態的聯繫，以確保雙方在專案開發組態上的配合。而在組織架構上有矩陣式、功能式、專案式等等 [2][16][17]。作者建議一物件導向之抽象化專案組織架構如[圖二]。對於小規模軟體組織而言，此架構為專案式與功能式組織的混合。此外，在 CMMI 的 Project Monitor & Control 流程領域中提及，專案管理中除將關鍵人員找出並指示工作職掌(responsibility)之外，需有適當的職權(authority)才能確保人員之執行度，及專案之執行能力[SEI, 2002]。人員在專案管理中扮演最重要角色之一，因此對於人員執行力與執行度的要求是 CMMI 及 People CMM 重要的規範之一 [22][24]。沒有賦予該職掌適當的職權來配合，常會有過程中人員執行度不足的情形 [10]。所以對於人員建制上，只定義角色與職掌是不夠的。例如，由於無權管理技術人員，客服人員往往無法掌握客戶問題表單之完成度。例如，組織管理階層可視問題表單之質(completeness and customer satisfaction)與量(number of forms, in terms of problems processed)為客服人員與技術人員之績效參考，且表單上有客服人員與技術人員之簽名。又例如，軟體組態人員須賦予職掌上相對的職權，如：變更需求提報審核、代理專案經理之職權、及文件品質控制並提報文件維護者之獎懲建議。因此，軟體組織之職掌與職權應同時在組織架構上定義。



圖二: 建議的抽象化專案組織架構(本研究整理)

然而小規模的軟體組織，員工經常扮演多重角色，或扮演相同角色但同時進行不同專案。針對此情況，屬於組織層面的企業資源規劃人員(通常由資深管理階層擔任)則扮演著相當重要的角色，負責專案間資源的協調，及人員工作的分配。[圖三]即為軟體組織將[圖二]的組織層級架構具體化來管理各專案，以期能消弭各別專案規劃看似可行但實際上卻無法做到的問題，進而確保各專案之執行度。

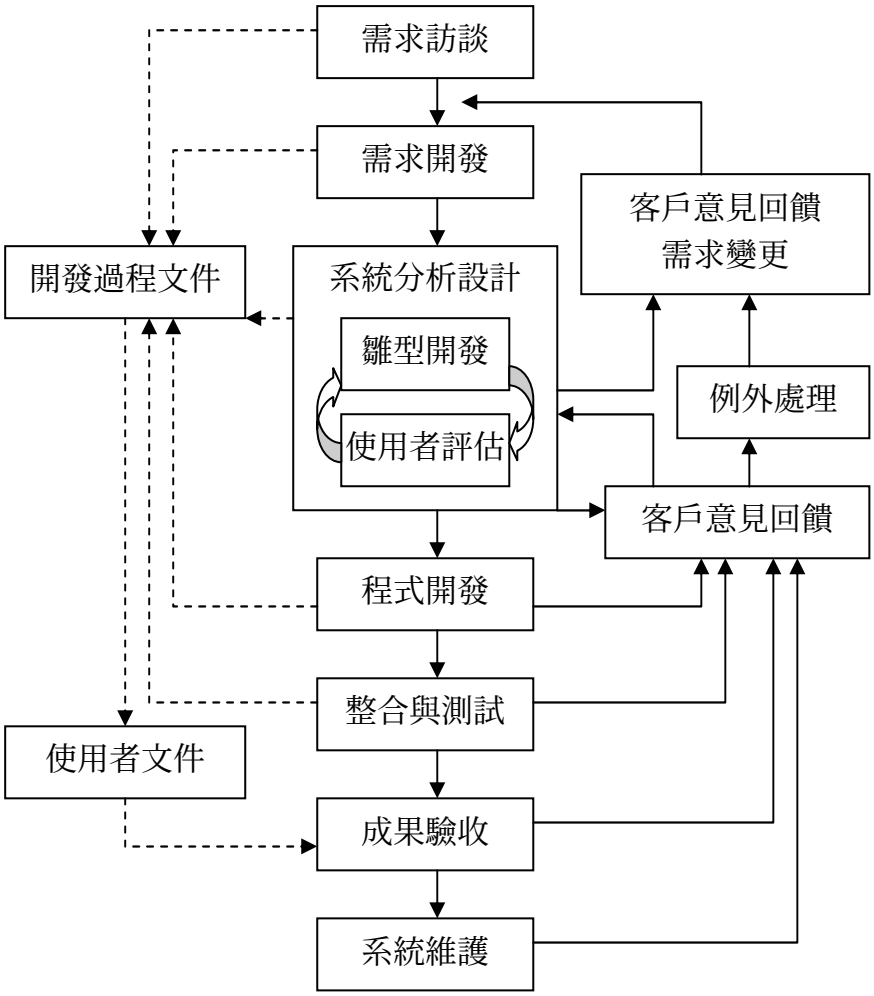


圖三: 由組織層級(企業資源規劃)來管理各個同時或非同時專案(本研究製)

參. 專案開發模式

關於專案之開發，有很多模式可供參考 [1][12]。以結構化快速雛型(Structured Rapid Prototyping)的方式來說，對於整個專案，自規劃完成後大致上分為需求訪談、需求分析、系統分析、系統設計、程式開發、測試與驗收、客戶驗收、客戶教育訓練及輔導上線、及系統維護服務等過程。大體而言，此開發過程中普遍皆有遞迴現象，以收同步及循序工程之雙效。特別是在分析設計階段，以確保未來產品的正確性。根據能力成熟度整合模式(CMMI)，在專案開發的每一階段皆有流程產物(Work products)，作者以為流程產物中的文件特別重要，因此軟體生命週期或專案開發過程都須和文件管理結合。圖片[圖四] 為本研究對結構化快速雛型軟體開發模式必須和文件管理結合的詮釋。

從[圖四]得知，專案開發模式應包含了流程文件的掌握，如此使得在開發過程中程式設計師對於軟體內容與搭配的文件能有效率地掌握。此外，一般而言，流程文件中的使用者文件往往是在軟體完成後，再依系統已開發出來的功能來加以定義的。然而如此作法有兩個缺失: (1)不易確定是否將客戶需求的初衷確實反映至文件上，亦不易驗證客戶需求的實踐度，及(2)導至日後文件版本與軟體版本的不搭配。軟體內容時常變更，但文件則常因管理階層沒要求或變更太過於煩瑣而忽略掉，或者是虛應故事，以致於文件與軟體內容之一致性愈加低落。因此，文件管理在專案開發過程中即須落實。



圖四: 作者對結構化快速雛型軟體開發模式必須和文件管理結合的詮釋

軟體開發的特色之一，即客戶需求時時變更。變更的產生有時因為是需求分析不當，有時則因為客戶端的客觀環境因素(例如，在一個長期性專案中，需求將會隨客戶的情況或供應鍊上的變更而變更)。因此縱使客戶需求在使用者試驗階段達成共識 (即[圖四]的使用者試驗區塊)，在開發過程下游階段時，需求仍有可能有影響性的變動而導致必須重新分析該需求。針對此情況，軟體組織提供需求變更例外處理(即[圖四]的例外處理區塊)。在考慮進度與可運用的資源，跟客戶進行溝通，以達到同時產品確實反應實際需求及反映成本的目的。以下部分即針對圖[三]的各個階段步驟逐一作大略的詮釋。

1. 需求訪談

一般而言，需求訪談為專案開發過程中最重要的一環。根據組織架構，指派一專責團隊，由專案經理帶領系統設計分析師與客戶單位進行結構式訪談(Structured Interview)。此目的在於利用結構化談話：(1)將客戶的需求描述加以結構化以利於需求的分析與編譯，(2)建立需求間之相依或順序關係(即：需求分解)，並方便下階段的工作分解圖(Work Breakdown Structure)之建立與驗證，(3)藉由結構化分析和 affinity

diagram、hierarchy tree 等之運用而了解每個需求的來龍去脈及邏輯，以便確認在編譯後的功能需求之正確性(Validity)。除了結構化訪談，系統分析師另外的重要工作則是:(1)將需求初步過濾以結合同質性的需求，(2)分解過於龐大的需求，及(3)將過於模糊的需求描述予以特徵化(Specialized and characterized)。

本專案在經過本階段後，客戶需求將被定義。在本步驟中所衍生的抽象化流程產物(work products)，例如:

R1. 訪談記錄文件或檔案
R2. 擬電腦化的原表單蒐集及客戶畫面設計
R3. 需求分解報告 (Requirement Affinity Analysis)
R4. 客戶需求敘述 (Customer Requirement List)
R5. 產品確認條件 (The Constraints of Validation)

2. 需求分析

本步驟之主要目的是將客戶需求(R3)轉變成功能需求。在軟體品質管理中，需求分析扮演著關鍵性的影響，並且 CMMI 將之獨立出來而成一流程領域。作者以為此步驟可為全面品質管理中品質機能展開之運用。在品質機能展開過程中，品質屋(House of Quality [20]) 能確切地將客戶需求轉變成技術性的功能需求及產品需求。再則，品質屋為一跨階段(分析與設計兩階段)的手法，其最終目的在於將白話的客戶需求描述變成技術性的系統功能、API meta、資料型態、資料表、及模組等等，以便於下階段的程式開發。系統範圍及與外部系統間(包含使用者)之資訊交流內容及模式(Information context)也隨著需求之功能化及技術化而成形。

在此階段，系統設計分析師持續和客戶進行訪談以進一步了解系統的工作及資訊流程，並透過上階段的系統環境圖，進一步開發出高階系統流程分析圖。本專案在經過本階段後，功能需求及產品需求將被定義。在本階段中所衍生的主要之抽象化流程產物，例如:

RD1. 訪談記錄文件或檔案
RD2. 變更申請
RD3. 變更通報及過程記錄
RD4. 品質屋 (Houses of Quality)
RD5. 系統環境分析 (Context Diagram)
RD6. 功能需求清單卡片化(Functional Requirements)或使用個案分析
RD7. 高階資訊流程分析 (Data Flow Diagram Level 0)

一旦系統功能成形後，在本階段另一重要工作即是針對系統功能中所須變更的技術需求及專案執行需求，透過企業層級資源規劃（見圖(三)），來修改計劃(Project Plan)中的資源(尤其是程式開發人員專業知識部分)之重調配及時程的修正，以確保整個專案之執行能力(Ability to perform)與執行度(commitment)。關於相關的變更管理手法請參閱專案管理部分的變更管理。

此外，一旦需求建立後，在接下來的開發過程中必將遇到變更。此時需求因一再

修改或因規模較小而忽略作文件上同步更新，其可循性(Traceability)變得越困難。eXtreme Programming 中的需求卡片管理手法來確定需求之實踐及雙向追蹤之能力。而客戶也可根據需求卡片來一一確認客戶需求的執行度(Validation)。有關需求管理方法請參閱專案管理部分的需求管理。

3. 系統分析

根據 Hoffer 在其專書提到，在此階段共有三個主要工作: System ' s process modeling、logic modeling，及 data modeling [12]。值得一提的是，在開發系統實體關係模型(ERD/EERD)時，正規化為一非常重要的一環。根據以往的經驗，一旦系統實體關係成型並且進入設計及程式編碼階段，任何有關 ERD 或 EERD 的變更將造成對於專案相當程度的衝擊，尤其是當大量資料已輸入及畫面設計完成之後。因此系統實體關係模型之正確性與變更彈性將使變更的代價降至最少。

在本階段中，客戶訪談應持續進行，以再確認客戶需求以針對最新變更作早期快速反應，並且進一步確認與功能需求(來自上階段)之一致性，或透過組織的變更管理(見專案管理部分)。另外，本階段客戶訪談之另一項目的為確認客戶期望在系統中呈現的資料。本專案在經過本階段後，系統資料流程將被定義，並且開發出符合此系統及客戶期望顯示的資料之模型。在本階段中所衍生的主要抽象化流程產物，例如:

SA1. 變更申請
SA2. 變更通報及過程記錄
SA3. 修正後功能需求清單
SA4. 資料流程分析 (Level 1, Level 2, and Primitive Level)
SA5. 初步邏輯設計的虛擬程式碼 (Pseudo Code) 及針對邏輯流程的決策樹狀分析 (Decision Trees)
SA6. 針對本專案的系統資料模型 (ER/EER/Class Diagram Model)
SA7. 系統活動圖 (Activity Diagram)
SA8. 系統循序圖 (Sequence Diagram)

4. 系統設計

系統設計大體上包含使用者畫面設計、資料庫設計、和細部規格設計。本階段之一重點為系統雛型之開發。此系統雛型在此時非能完全被操作，然而客戶卻能從雛型獲得未來系統的輪廓，並作進一步的產品溝通，以確保客戶的原創需求之徹底實踐。如此與客戶不斷的進行產品界面的溝通與確認即為本階段所採用雛型式專案進行模式之原因。

另一個採用雛型式的原因，便是為了因應不斷的變更。經由初步設計成形後，專案組織將與客戶進行討論並且根據客戶對雛型的反應來修改。然而根據經驗，此時的需求開發尚未成熟(尤其是客戶本身對需求尚未成型)，即，在此時專案組織雖已完成定義的功能需求，但極可能因客戶新增及變更的需求而作相關的變更處理。因此變更處理及如何維護雛型品質(端賴於變更管理之有效實施)在此時是工作重點之一。關於變更處理，請參閱[圖四]的軟體開發模式變更處理程序部分，及專案管理中的變更管

理部分。

專案在經過本階段後，系統資料流程將被定義，並且開發出符合此系統及客戶期望顯示的資料之模型。在本階段中所衍生的主要抽象化流程產物，例如:

SD1. 訪談記錄文件或檔案
SD2. 變更申請
SD3. 變更通報及過程記錄
SD4. 修正後功能需求清單
SD5. 資料庫建制表集
SD6. 資料庫安全系統(Replication system)設計
SD7. GUI 模擬
SD8. 系統、元件、及模組規格書
SD9. 修正後工作分解圖

5. 程式開發

程式水準與撰寫品質關係到整體系統運作的成敗。任何一個潛在的規模程式錯誤都必須在這時期充分找出並除錯。針對由上階段產生的模組化系統設計，再根據 work breakdown structure (WBS)，來展開程式開發工作。

根據過去經驗，維護在系統時最困難的一工作之一為程式追蹤解讀。尤其追蹤解讀非本公司所開發的程式碼。針對這一問題，專案組織應重視及執行程式碼註解制度。並在程式碼中建立責任標示，以便快速追蹤程式來源、設計邏輯、及負責撰寫及最近修改之人員。另外，在程式開發過程常會使用現有的軟體元件(API)。適當的 API 系統實為知識管理之一種手法，也是成熟的軟體業者應具備的特色。

版本控制為本階段之管理重點，也為軟體品質管理中必要的手法。對於專案開發的版本控制，作者建議採取資料庫管理中 ACID (Atomicity, Consistency, Isolation, and Durability)原則 [9]，與組態管理中 Configuration Management II，簡稱 CMII，之管理手法 [13]。有關版本控制詳細內容，請閱下一部分的專案管理之組態管理及版本控制部分。

經過本階段，系統功能將完全被實作，並達到可使用之地步。在本階段中所衍生的抽象化流程產物，例如:

D1. 變更申請
D2. 變更通報及過程記錄
D3. 程式版本控制查詢記錄
D4. 程式改善提案追蹤記錄
D5. 階段內測試記錄
D6. 程式註解及 API 系統

6. 整合與測試

模組、系統整合在軟體專案開發過程中扮演著關鍵性的角色。Kan 在其專書提到

因這步驟所導致或發現的錯誤(Fault)即佔全部開發過程的 48% [15]。在小規模軟體組織環境下，由於成本及人員數目的限制，測試往往由系統開發人員兼任，如此常導致測試盲點，即錯誤不易察覺，尤其是在邏輯部分。而原因即是相同的人對於相同程式有著的固定思考邏輯所致。因此，專案組織應要求由不同單位或人員來執行這項工作，以確保測試的績效(Defect removal effectiveness)進而確保軟體的品質。

除了偵錯，本階段另一工作重點是執行因除錯而需要的程式變更。由於每個規模錯誤(或幾個同質性錯誤為一單位)將成為問題提報並啟動變更管理及稍後的版本控制，但若將大小問題皆提報做為變更處理則是浪費資源及時間。因此，問題確認這項工作對於有效率地運用企業資源而言極為重要。關於變更管理，專案組織應有一套標準作業程序(Standard Operation Procedure, SOP)並將之制度化，並且對於每個變更，都有文件可循。請參閱專案管理部分中變更管理之說明。

經過本階段，系統運作將完全被驗證，系統產品並達到可交付給客戶之狀態。在本階段中所產生的抽象化流程產物，例如為:

IT1. 變更申請
IT2. 變更通報及過程記錄
IT3. 程式版本控制查詢記錄
IT4. 新增或修改之功能需求
IT5. 測試計劃及測試個案集
IT6. 系統測試記錄
IT7. 開發人員評鑑資料
IT8. 使用者文件、產品說明及規格

7. 客戶驗收、教育訓練、及輔導上線

客戶將針對其使用模式來作驗收前的測試。一旦測試不合格，專案組織將針對不完善處進行變更需求來修改系統，直到測試合格。一旦測試合格，則進行系統移交(Deliverable)。為確保軟體品質之全面性，當系統完成交付後，專案組織應提供下列抽象化表述之教育訓練:

1. 系統使用流程介紹
2. 系統規格說明
3. 線上操作練習
4. 線上操作教練文件

在系統規格說明和線上操作教練文件方面，系統規格說明為以後開放性系統擴充及改善之用，以儘量減少對原開發廠商之依賴度。而線上操作教練文件則是針對本專案產品在長久使用以後該系統使用單位之新進人員得以進行獨立自我訓練用(訓練使用本軟體產品)。經過本階段後，系統產品交付給客戶，並提供相當的使用者教育訓練。在本階段中所衍生的抽象化流程產物包括:

DER1. 變更申請
DER2. 變更通報及過程記錄
DER3. 版本控制查詢記錄

DER4. 修正後使用者文件、產品說明及規格
DER5. 線上操作練習教材
DER6. 結案報告書

8. 系統維護與服務

為達全面品質管理訴求，在專案產品完成後之有限期限內，專案組織應負責系統維護之任務。此階段之主要工作為系統 Frequently Asked Question(FAQ)及知識管理之建立。一般而言，專案組織皆提供專案完成後一定期間的產品服務及使用者教育上線保證。在這保證期間內，對於使用手冊等相關客戶擁有之文件所無法回答的問題，專案組織都將儘力協助解決，並安排專業人員至系統現場(On site)進行維護及解說。

表格[表一]為上述專案開發過程中產出或導出(derived)的關係圖。本研究以為，建立起文件之間的對應及來龍去脈將有助於文件管理制度化，並且提升文件變更分析及追蹤的能力。因此作者建議在文件實體上，或於文件索引(document reference)上標示 Contextual Relationships。例如，關係: R1(訪談記錄文件或檔案)→R4(客戶需求敘述)則在該客戶需求敘述標示是出自於某個特定的訪談記錄文件版本或檔案。

表一: 本研究認為之的專案開發過程中文件之關係圖 (請參照文中之編號)

Phase	Contextual Relationships
需求訪談	{R1,R2,R3}→R4;R4→R5;
需求分析	{R4,R5}→{RD1,RD3};{R5,RD1}→{RD2,RD3}; {R5,RD3}→{RD4,RD5,RD6};
系統分析	RD5→{SA1,SA3};{SA1}→{SA2,SA3};{R5,SA3}→SA4; {R5,SA3,SA4}→SA5;{R5,SA3,SA4}→{SA6,SA7};{R5,SA5,SA7}→SA8
系統設計	{SD1,SD2}→{SD3,SD4};{R5,SD4,SA6}→SD5;{R5,SD5}→SD6; {R5,SD4}→{SD7,SD8};
程式開發	{SD4,D5}→D1;D1→{D2,D3,D4};{DER3,IT3,SD4,D3}→D6;
整合與測試	{R5,SD4,IT6}→{IT1,IT5};{R5,IT1}→{IT2,IT4};IT5→IT6; {IT2,IT6}→IT7;{R4,R5,SD4,IT4,SA6,SA7,SA8,SD6,SD8}→IT8
驗收與上線	DER1→{DER2,DER3};{IT8,DER1}→DER4; {R4,R5,SD4,IT4,SA6,SA7,SA8,SD6,SD8,DER4}→DER6;

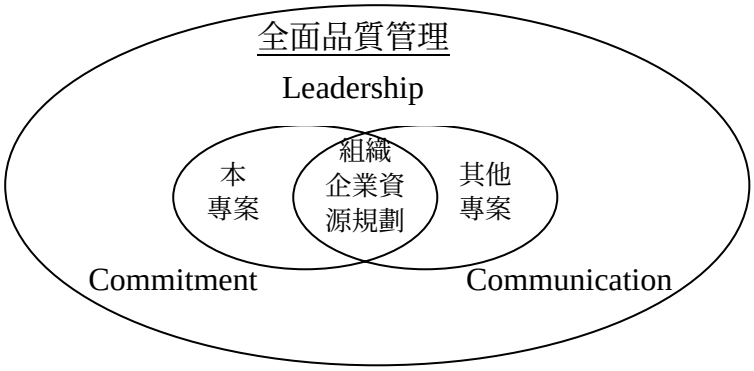
註: A→B 表示由文件 A 導出文件 B，或文件 B 是根據文件 A

四、專案管理方法

專案之執行力有賴於專案的制度化管理，其內容包括 CMM 層級二之流程領域，亦即組態管理、供應商管理、專案進度控管、與需求管理。針對組織的每一個專案，作者建議一套管理方法。這套專案管理方法主要是在兩方面: (1)制度化的開發過程管理(包括前述的組織架構及專案開發模型、及本部分的組態管理及專案控管)、與(2)文件的管理(包含了產品及流程管理所產生的文件、及對系統描述的傳統文件或物件導向

的 UML 圖集)。

而在企業組織層級方面，導入全面品質管理(TQM) [11][20]。作者以為全面品質管理可分為兩方面: (1)對軟體產品整個生命週期的品質保證，及(2)整個企業對各個專案產品品質的共識與支援。前者可由適當的開發過程模式來達成。而後者說明了軟體品質除了藉著(1)來達成外，面對專案開發過程中不斷的變更，各個專案層級也需要企業組織層級的支援與動態協調。因此，根據 Oakland 在全面品質管理中的論述 [20]，以及上述的看法，TQM 在專案管理上可有如[圖五]的詮釋：



圖五: 本研究對於專案管理在全面品質管理的詮釋

從上面的圖中以及過去的經驗告訴我們，一個永續經營的軟體企業，其當下所開發的各個專案實際上無法獨立要求及管理。因此軟體組織特別注意到專案層級間的需求整合與管理。專案需求的整合與管理在個別專案規劃階段即進行評估，並在專案控管上，透過組織會議(請參閱以下的專案控管部分)來作動態的調度與再分配以維護專案規劃上的協議。而要使如此的整合性產品與流程管理有成效則端賴一套變更影響分析機制，並加以制度化。

為達成全面品管目的，針對專案獨立開發的軟體組織之專案管理方面，作者建議以下的項目，並依照 CMMI 層級二的模式，落實下列專案流程領域之特定目標(Special Goal)及特定執行方法(Specific Practices)，並在每個特定執行方法中確定實施一般執行方法(Generic Practices)準則，以達到層級二的一般目標(Generic Goal):專案進度控管、組態管理，包括變更管理、文件管理、與版本控制、及需求管理。

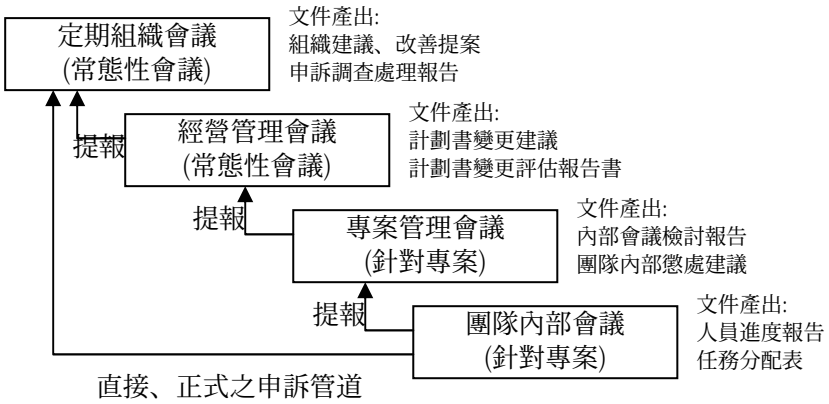
1. 專案進度控管

在專案控管主要是經由各層級的會議來了解並管理專案進度。其組織架構如[圖六]所示。

例如，在專案的各研發團隊於稍早召開團隊內部會議。管理階層於稍後召開專案管理會議，由專案召集人聽取並審閱專案經理、企業資源規劃、變更委員會的進度分析報告。針對其控管的內容，在此作一描述:

1. 根據規劃出來的時程表來監督進度: 按照階段性任務的時程進行檢驗。召開專案團隊內部會議以檢討工作內容並確認應做項目。若遇到進度落後或工作內容品質低落，除檢討原因，並於會後發文報告專案經理以讓組織及管理階

層即早應變。



圖六: 本研究所提之軟體組織對於透過會議來掌握專案相關進度的架構

2. 監督目前專案所耗費的金額及資源使用情況: 於經營管理會議中檢視各專案資源運用情形，至於變更部分則和企業資源規劃人員協調。並由專案召集人檢視預算運用狀況。
3. 監督應完成的產品需求: 根據時程的規劃，於團隊內部會議中驗收階段性雛型產品。
4. 監督人員工作情況: 除了平日觀察, 利用團隊內部會議, 由小組長(project lead)負責記錄人員工作進度及完成品質，並在專案管理會議中提報給專案經理做為日後績效獎金配額之依據。專案經理則負責監督小組長是否有確實記錄。並有一正式管道供員工申訴用。

此外，專案執行力端賴專案開發過程中不斷的控管，並藉由專案內及組織層級會議的召開來執行控管。而運行一個有效率的會議則影響專案控管的品質。關於如何召開有效率的會議請參閱相關書籍或論述。

2. 文件管理

本研究以為，組態管理主要內容為文件管理、變更管理、與版本控制。在軟體組態管理中，文件管理為專案管理中最重要的工作項目之一。文件的有效管理可以:(1)幫助企業組織了雜亂無章的流程並有統一的標準與格式，(2)簡化並加速員工之資訊及經驗交換，尤其是人員易動時技術的交接，(3)使得企業和客戶有效率地掌握專案進展, 並作快速之決策支援, (4)增進日後對於類似專案之開發(包括程式開發部分所需要的 API 文件)有具體有形之經驗可參考，亦即知識管理的落實方式之一，(5)幫助組織在日常專案管理及突發的危機應變有了第一時間的標準應變措施(教戰守則)。

隨著能力成熟度整合模式的推展，組織應有一套組織文件管理系統(Centralized, Replicated Document Master)並可用於個別專案之管理。此文件管理系統能清楚地呈現文件的完成日期(但還不見得可以使用)、即將有效日期(經稽核單位驗證後可使用)、使用日期、版本、及版本歷史記錄；並清楚地呈現文件的所有使用者、修訂者、及文件開發者。一般而言，此系統應包括下述功能:

- 版本管理: 為防止文件版本混亂使用，及確保使用到正確的版本，本系統在文件的說明部分(Meta Data)建置了修改中標示、完成日期、即將有效日期(經稽核單位驗證後)、使用日期、版本、及版本歷史記錄。當文件完成後，還不見得可以使用及引用，必須經過核驗程序方可被引用。

- 除了版本編碼，針對版本管理，組織定義文件的狀態，以三個狀態為例:修改、可引用、和歸檔。並有適當的角色(參見關鍵人員責任分級部分)來改變文件的狀態。當文件為修改狀態時，而此時文件為唯讀並且在該時點該版文件無法被引用。當文件為可引用狀態時則該時點該版文件可被引用。當文件為歸檔狀態時則表示有新一版本文件已達可引用狀態。對於系統而言，文件只能有一版本是在可引用狀態。歸檔文件意指可隨時被讀取。

- 關鍵人員責任分級制: 依照資訊安全中 RBAC(Role Based Access Control)的原理，將接觸文件的人員依接觸文件的情況而分級。例如分成下列角色:使用者、修定者、開發者、及代理人員。例如，使用者只能讀取文件但不能作修改。且每份文件有不同的使用者集合。代理人員是針對上述角色不在時(尤其長時間離開如出差)能有責任代理的功能，且文件能依然可控管與編輯。

稍後在程式版本控制部分採用文件管理類似的作法。請參閱該部分以深入了解組織對於版本控制的手法。至於文件品質:除了要求輸出驗證及審核過程，上述的文件管理可有效提升文件使用品質。

3. 變更管理及版本控制

在軟體開發過程中，常是多人同時更新某程式區塊，或多人共用同一程式碼以開發各自的程式。另在開發過程中，因修改、偵錯、除錯所造成的變更不勝枚舉。在這情況下，不斷的軟體變更在多人開發環境中容易導致程式混亂度與複雜度提高而致軟體品質不良。組態管理系統即是針對此一普遍性問題而產生。在建立此系統時，本研究建議應用資料庫管理中的 ACID (Atomicity, Consistency, Isolation, and Durability [9])原則於變更管理版本控制中，加上文件管理中所提及的版本管理及 RBAC 的責任分級制，和 CMII 中關於變更管理的運用，再配合能力成熟度整合模式中的組態管理流程領域之特定實行方法，發展出一套適當的變更管理與版本控制機制。有關建議的系統中的功能、ACID 之應用、程式碼狀態分級、及程式關鍵人員角色分級制分述如下。

- ACID 原則在版本控制與變更管理的延申:
    1. Atomicity: 每次的修改只有兩種結果:成功並替代了原版本程式(Commit)，或者是取消並且還原原版本程式(Rollback)。
    2. Consistency: 此原則確保系統所有程式碼都在單一狀態；即修改中、可使用、及歸檔狀態為 mutually exclusive。
    3. Isolation: 每一次的程式修改將導致該類別或檔案被鎖定而成唯讀狀態。此狀態直到修改 commit 或 rollback 為止。此原則並確保同一程式碼在同一時間，任何地點只能有一人在修改。
    4. Durability: 確定每次成功的修改都有效並永久記錄在資料庫中。

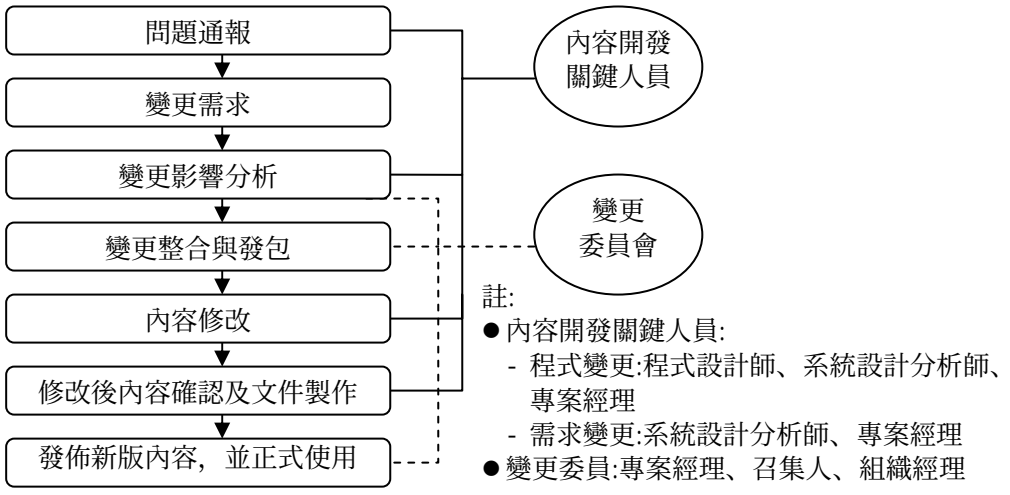
- 程式碼狀態分級:
  1. 修改中:處於修改中的程式碼為唯讀, 並且所有關鍵人員不得同時修改該段程式。若在時間壓力情形下, 可透過變更程序委員會(參閱變更管理部分)決定其影響狀況(Impact analysis)並可能合併為同一變更項目以同時進行並且為同一修改版。
  2. 可使用:一旦程式更改完成並經過核驗即可成為可使用狀態。當程式碼有物件聚合(aggregation/composition)、繼承(inheritance and/or interface)、多形(polymorphism)、或鏈性呼叫(chaining)則必須註明使用或呼叫的物件、類別、或程式碼的使用版本。
  3. 歸檔版: 一旦程式更改完成並經過核驗, 原程式碼即為前一版並歸檔。歸檔版的程式碼不得拿來使用, 只拿來視察或過去查驗時使用。

- 程式關鍵人員角色分級制則參考上述關鍵人員責任分級部分。此外, 對同一程式碼而言, 其狀態在同一時間, 任何地點只能為單一狀態(Mutually exclusive), 即是處理變更及版本控制的原理。

4. 變更管理程序與 CMII

變更開始於問題的發現。當有問題呈報時, 工程師必須先自行檢視該問題是否為規模問題。如是, 則提出變更申請交付組態工程師。然後由變更委員會逕自, 或指派適當的軟體工程師、程式設計師、或系統設計分析師進行該變更需求之影響分析, 找出須要更動的其他程式區塊。接下來委員會會將變更需求作判斷、歸類並且同質需求統一發包, 以節省人力及時間。程式設計師根據需求進行修改, 完成後由委員會或其指派測試人員檢驗。完成檢驗及成為該程式碼最新版本。[圖七]即為作者所建議的抽象化軟體組態變更程序。

為了針對員工有“一旦報請變更, 則等於是跟公司自曝自己的錯失或能力不夠”的迷失(此迷思常導致程式隱藏性錯誤之增加), 組織應有制度化的鼓勵措施, 例如一旦變更成立(透過變更委員會), 組織即對該提報人員作適當的記點獎敘, 用以消除隱藏性錯誤, 以增進軟體產品品質。



圖七: 抽象化的變更程序

5. 需求管理

eXtreme Programming (或稱為極致)為一種新興的軟體開發模式[2][4][19]。作者建議運用其中的故事卡片(story card)來管理需求，即下述所謂的卡片需求管理，理由是透過需求卡片化，管理簡易，且不失對原創需求之追蹤能力。由於需求為文件的一種，故上述的文件管理、版本控制、及變更管理可運用至本管理項目。因此在 eXtreme Programming 之協助下，以為專案層級組織對於需求管理有下列之工作重點:

1. 了解需求: 了解需求為專案管理及軟體品質之第一訴求。產品是否符合期望常繫於開發團隊對需求之了解與否。在專案進行初期，除了在專案投標所獲得的需求外，開發團隊需與客戶就需求收集進行會議討論。然而除了需求的完全蒐集外，完全了解這些來自客戶的描述並轉變為功能需求亦是重點工作。品質屋(House of Quality)為所採用的需求編譯的方式，從白話的客戶描述，到功能需求及虛擬程式碼皆能做逐一對照。
2. 對於需求達成的執行度: 本工作著重於找出功能需求間的依存性(Interdependencies)，並根據此建立工作分解圖(WBS)。再根據工作分解圖與人員建制表而適當地安排在時程表中，藉此掌握專案之執行度。另外，藉由需求的卡片管理，將編譯後的需求妥善管理。在專案整個開發過程即是卡片上的需求的驗證(Validation)。
3. 需求變更管理: 需求變更來自兩情況：(1)因主客觀因素，客戶改變初衷而增刪或修改需求，或(2)開發團隊改變原本解決方案或設計。針對需求的變更，本研究認為按照專案管理的變更管理方式(請參閱變更管理部分)。首先須調查需求變更所帶來的影響之分析。而需求變更管理之流程則如圖七所示，惟其『內容』部分根據 CMMI 具體化為客戶需求(Customer requirements)，產品需求(Product requirements)、或產品元件需求(Product-component requirements)[22]。
4. 雙向需求追蹤能力: 需求卡片管理除了確保需求的被實踐，還可用來追蹤需求在每一階段的完成情形。

伍、結論

在國內各地相繼成立軟體科學園區並向國際舞台邁進之際，軟體品質管理可說是軟體企業接下在來軟體工程，專案管理，或軟體品質領域中首重的工作之一。尤其當軟體企業準備進軍軟體國際市場時，如何安撫客戶已不再是專案開發過程的 corrective action 權宜之計。反之，加強組織本身對於各個專案的執行力、執行度之掌握將成為企業永續經營的必要條件，因為我們知道，在全面品質管理觀念中，執行力、執行度之掌握與否影響了軟體品質綜合指標，進而影響客戶對軟體產品及專案開發品質的整體滿意度。

昂貴的軟體工具及軟體自動化似乎並非管理專案開發的唯一的方式。作者從管理手法，及抽象化專案組織架構、專案開發模式、及 CMMI Level 2 的專案管理三方面探

討軟體組織如何掌握專案執行度、執行力，進而改善軟體品質，以作好預知性品管。抽象化專案組織架構如[圖二]及[圖三]，即每個專案產生、具體化一個相關之人事組織架構，但組織層級架構則不變(static)，並用來作專案間的動態協調。專案開發模式如[圖四]，尤其特別重視文件的定義及相互間的來龍去脈，即如[表一]之 contextual relationships，以作為日後追蹤、稽核，並作為類似專案之知識管理用。CMMI Level 2 的專案管理則包括專案控管如[圖六]，專案開發時的組態管理(包含文件管理、版本控制機制、變更管理如[圖七])，及需求管理)。

對軟體品質管理而言，能力成熟度整合模式不是企業改善的唯一方式。然而在結果預知性的合作模式下，它逐漸成為軟體供應鏈中各層級的客戶選擇委外單位或供應商之依據，或與之溝通之管道(protocol)。在軟體業者試圖掌握專案執行力與執行度的同時，逐漸導入制度化的流程管理模式如 CMMI，以改善組織及其流程的體質，進而縮短日後進行國際認證目標時的差距(Gap analysis)。

參考文獻

[1] 吳仁和、林信惠，*系統分析與設計理論與實務應用*，智勝圖書股份有限公司。

[2] 林信惠、黃明祥、黃文良，*軟體專案管理*，智勝圖書股份有限公司，2002。

[3] 資策會譯，能力成熟度整合模式 v1.02，財團法人資訓工業策進會出版，2003。

[4] Beck, K., *eXtreme Programming eXplained*., Addison Wesley, Upper Saddle River: New Jersey, 2000.

[5] Belout, A. and Gauvreau, C., "Factors influencing project success: the impact of human resource management," *International Journal of Project Management*, Volume: 22, Issue: 1, January, 2004: pp. 1-11.

[6] Bootstrap Project Team, "Bootstrap: Europe's Assessment Method". *IEEE Software*. May 1993: pp. 93-95, 1993.

[7] Chatzoglou, P.D., and Macaulay, L.A., "The Importance of Human Factors in Planning the Requirements Capture Stage of a Project", *International Journal of Project Management*, Vol. 15, Issue 1, February, 1997: pp. 39-53.

[8] DoD, *Military Standard of Software Development and Documentation*, US Dept. of Defense, September 1994.

[9] Elmasri, R. and Navathe, S.B., *Fundamentals of Database Management Systems*, Addison-Wesley, Upper Saddle River: New Jersey, 2000.

[10] Elonen, S. and Artto, K.A., "Problems in Managing Internal Development Projects in Multi-Project Environments", *International Journal of Project Management*, Volume: 21, Issue: 6, August, 2003: pp. 395-402.

[11] Goetsch D.L. and Davis S., *Quality Management: Introduction to Total Quality Management for Production, Processing, and Services*, International Inc, New Jersey, 2003.

[12] Hoffer, J.A., et al., *Modern Systems Analysis and Design*, Prentice-Hall International

Inc, New Jersey, 2002.

[13] Institute of Configuration Management, *Configuration Management II*, the ICM training material, Phoenix: Arizona, 1999.

[14] ISO, EIA/ ISO15507, Online WWW: http://www.iso.org/iso/en/ ISOOnline.frontpage, 2003.

[15] Kan, H.S., *Metrics and Models in Software Quality Engineering*, Addison Wesley, Upper Saddle River: New Jersey, 2002.

[16] Knight, K, editors. *Matrix management*. Gower Press, Aldershot, UK, 1977.

[17] Kuprenas, J.A., "Implementation and Performance of a Matrix Organization Structure", *International Journal of Project Management*, Volume: 21, Issue: 1, January, 2003: pp. 51-62.

[18] Lister, T., "Hallucinations at 37,000 feet", *IEEE Software*, Vol. 15, No. 3, May-June 1998: pp105 -107.

[19] Nawrocki, J., et al., "Toward Maturity Model for eXtreme Programming", IEEE, 2001.

[20] Oakland, J.S., *Total Quality Management: The Management of Change through Process Improvement*, Oxford: Butterworth Heinemann, 1993.

[21] Rus, I. and Lindvall, M., Knowledge Management in Software Engineering, *IEEE Software*, May/June 2002: pp26-38,

[22] Software Engineering Institute (SEI), *CMMI-SE/SW/IPPD/SS Staged Representation* v1.0, Carnegie Mellon University Press: PA, 2002.

[23] Software Engineering Institute (SEI), *Standard CMMI Appraisal Method for Process Improvement v1.0*, Carnegie Mellon University Press: PA, 2002.

[24] Software Engineering Institute (SEI), *People Capability Maturity Model (People CMM) v2.0*, Carnegie Mellon University Press: PA, 2001

[25] Zubrow, D., "Current Trends in the Adoption of the CMMI Product Suite"**,** *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, Nov. 2003: pp126 -129.