行政院國家科學委員會專題研究計畫 成果報告

適用於企業計算之軟體工廠模式(III)
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 99-2221-E-029-017-
執 行 期 間 ：99 年 08 月 01 日至 100 年 07 月 31 日
執 行 單 位 ：東海大學資訊工程學系

計 畫 主 持 人：周忠信
共 同 主 持 人：鄭有進
計畫參與人員：碩士班研究生-兼任助理人員：王京郁
碩士班研究生-兼任助理人員：張哲綜
碩士班研究生-兼任助理人員：陳宜蓁
博士班研究生-兼任助理人員：徐天送

報 告 附 件 ：出席國際會議研究心得報告及發表論文

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 100 年 10 月 27 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告 □期中進度報告

適用於企業計算之軟體工廠模式-III

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC 99-2221-E-029-017-
執行期間：　99 年　8 月　1 日至　100 年　7 月 31 日

執行機構及系所：東海大學資訊工程學系 AAJC 實驗室

計畫主持人：周忠信 東海大學資訊工程學系 副教授
共同主持人：鄭有進 國立台北科技大學軟體發展研究中心教授
計畫參與人員：徐天送　國立台北科技大學資訊工程系博士生
張哲綜、陳宜蓁、王京郁 東海大學資訊工程學系 研究生

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本計畫除繳交成果報告外，另須繳交以下出國心得報告：
□赴國外出差或研習心得報告
□赴大陸地區出差或研習心得報告
■出席國際學術會議心得報告
□國際合作研究計畫國外研究報告

處理方式：除列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

中　華　民　國　　100 年　　10 月　5 日

一、中文摘要

軟體工廠的生產模式早在 70 年代就被提出，但軟體工廠的概念卻未獲得認同，究其原因在於軟體的應用領域既廣泛而複雜，而硬體工廠生產機制卻僅能生產一定類型的產品。本計劃提出虛擬軟體的概念，並基於虛擬軟體進一步提出軟體生產線模型，使得類似工廠生產模式的軟體工廠能夠實際存在，進而提升軟體生產速度，並確保軟體品質。

關鍵字：軟體工程、軟體工廠、虛擬軟體、軟體生產線

Abstract

As early as in the 1970's, the notion of software factory that treats software development as a kind of factory production had been proposed. However, software factory has not yet been accepted by the software development community. The major reason is that factory model is restricted to produce fix-type products. In this research based on pseudo software, a corresponding software product line model is proposed. With this model, a software factory resembling hardware factory model is possible to be build.

Keywords：software engineering, software factory, software product line, pseudo software

二、前言

本計劃中所預計發展的軟體工廠模式，目標是希望可以模仿硬體生產的工廠運作機制。以往硬體廠商長久以近似工程技術來設立一個產品生產線，用以一群生產外觀、功能相近的產品，並透過大量生產(Mass Production)來降低生產成本，這是屬於一種規模經濟。然而，硬體廠商為符合市場需求，相同的產品再也無法滿足不同族群的顧客，而逐漸發展出少量多樣的生產模式(High-Mix, Low-Volume Production, HMLV Production)，透過市場區隔來提升產品附加價值而獲利，這是一種範籌經濟。而少量多樣的生產模式，其關鍵在於生產的硬體產品，原則上還是屬於同一種類型的產品。硬體廠商在生產線上對於同一類型的產品，也可說是屬於同一種樣式，儘管各有不同，但生產者可以根據明確的物料清單 BOM 與生產說明，即能實際生產出這些少量多樣的產品。因此，我們模仿這樣的概念，於第一年構想的 SoftBOM，亦即軟體的物料清單，還須配合一個可以在需求面上描述預計開發軟體的表達方式。儘管使用案例(Use case)是一個非常理想的需求定義機制，然而僅利用使用案例顯然也無法直接與 SoftBOM 銜接，然後提交軟體工廠付諸生產。本計劃所發展的虛擬軟體概念，基本上可以解決上述所提及的困難。當虛擬軟體概念得以落實後，對應 SoftBOM 精神的需求表達方式也才可以落實。因此，就像硬體工廠一樣，當一個軟體工廠可以事先發展出其 SoftBOM，而需求發展者則利用虛擬軟體中對應該工廠的 SoftBOM 來定義軟體需求時，這個虛擬軟體描述的軟體需求，就可以被該軟體工廠接受作為生產依據。而最重要的是，這樣的生產概念具有以下幾個優點：

- 軟體生產者與需求提供者間，在生產過程中的耦合度大量降低。
- 需求錯誤與變更大量降低。
- 由於需求定義時已採用自己工廠的 SoftBOM，可以提升軟體生產速度，並確保軟體品質。
- 類似硬體工廠生產模式的軟體工廠得以實際存在。

本計劃今年發展一個適用於軟體工廠之達交預估(Delivery Forecasting)系統。達交預估系統在軟體工廠管理中，扮演一個相當需要且重要的角色。當軟體工廠在接受以虛擬軟體需求模型作為訂單時，必須如同硬體工廠一樣，能夠適時地回覆客戶待開發軟體的預估交期與成本。在軟體工廠中，達交預估最好能同時包括軟體工廠的派工與生產排程建議，如此才能作為工廠真正生產時的依據。因此，本計劃在發展達交預估系統所需之軟體工廠的預測模型，再根據所有可用資源與限制條件，預估達交的日期與成本。

三、研究目的

今年的計劃以發展一個可以模擬軟體工廠實際運行的達交預估系統，並持續開發軟體工廠所需之相關工具，並依據前兩年所發展的虛擬軟體與 SoftBOM 模型為基礎，發展出一種軟體模仿硬體工廠的軟體工廠生產方式，主要的研究目標可分成下列兩點：

(1) 建立預測模型，並發展軟體工廠所需之達交預估系統：
本計劃根據工廠可用資源與限制條件作為模型的主要輸入參數，透過預測模型來預估達交的交期與成本，並進一步發展出適合軟體工廠所需之達交預估系統。

(2) 持續開發軟體工廠所需相關工具：
以 Android 應用程式為對象，持續發展一個軟體工廠所需之支援工具，令使用者能用以建立虛擬軟體之需求模型。

四、研究方法

4.1 模仿硬體生產模式之軟體工廠模型

本計劃所提出的軟體工廠模型，是從需求規格開始，直到產生最終軟體之間的整個運作流程。為發展出類似硬體工廠之生產模式之軟體生產線模型，我們提出之的 SoftBOM 視為軟體生產線研究中扮演關鍵角色的軟體資產(software assets)，用以在軟體生產線中重複使用。

我們運用虛擬軟體之需求模型來塑模所要發展軟體之軟體需求，透過虛擬軟體使得軟體開發團隊能在早期階段得以確認軟體需求，並解決 IKIWISI 的困擾。而每一個 SoftBOM 均可分為 SoftBOM 虛擬軟體之需求模型及 SoftBOM 範例程式兩個部份。前者運用虛擬軟體來塑模 SoftBOM 的需求模型，以 Scene 的方式具體描述軟體需求的使用情境、Interface Component 描繪其 GUI 操作介面、Business Logic 描述其商業邏輯等方式，並在 Test Case 中記錄軟體通過使用者可接受度測試之所必須檢驗的測試案例。SoftBOM 範例程式則提供與 SoftBOM 實作相關之程式碼骨架、參考函式庫及元件等內容，並提供許多程式碼範例協助程式撰寫人員能夠正確且無誤地實作該 SoftBOM 的功能。

當虛擬軟體概念得以實現，並發展軟體工廠所需之各種 SoftBOM 後，基於上述所提及的軟體生產線模型，配合虛擬軟體需求模型，我們進一步提出軟體工廠生產模式。如圖 1 所示，我們使用虛擬軟體需求模型來蒐集、塑模所開發軟體的需求，並透過 Pseudo Software 模型模擬及可操作的方式，讓軟體開發團隊得以在早期階段確認軟體需求。而需求塑模者(Requirements Modeler) 則根據軟體工廠所能支援的 SoftBOM 類型，發展目標軟體的虛擬軟體需求模型，如同硬體工廠只能生產某一生產類型的產品，我們所提出的軟體工廠模型中，需求規格必須符合該軟體工廠所提供的生產類型，才得以交由軟體工廠進行生產。

軟體生產線中的軟體核心資產則交由 SoftBOM 開發人員負責開發及維護，SoftBOM 開發人員在確立需求規格的生產類型後，可事先挑選合適的 SoftBOM 或發展出所需的 SoftBOM。若要發展一個新的 SoftBOM 模型，SoftBOM 開發人員必須先發展出該 SoftBOM 的虛擬軟體需求模型，並依據 SoftBOM 虛擬軟體需求模型實作出該 SoftBOM 的範例程式，提供程式撰寫人員在各別的軟體生產線中作為參考。最後，SoftBOM 開發人員將這些已開發完成的 SoftBOM 集中在 SoftBOM 資源庫中進行管理，使得這些軟體核心資產能夠在需求塑模階段中重複使用，以及作為軟體進入生產階段中重要的開發參考依據。

當軟體進入至生產階段時，依目標軟體所使用的 SoftBOM 開設眾多不同的生產線，每條生產線均由程式撰寫人員來負責實作 SoftBOM 虛擬軟體需求模型中所要

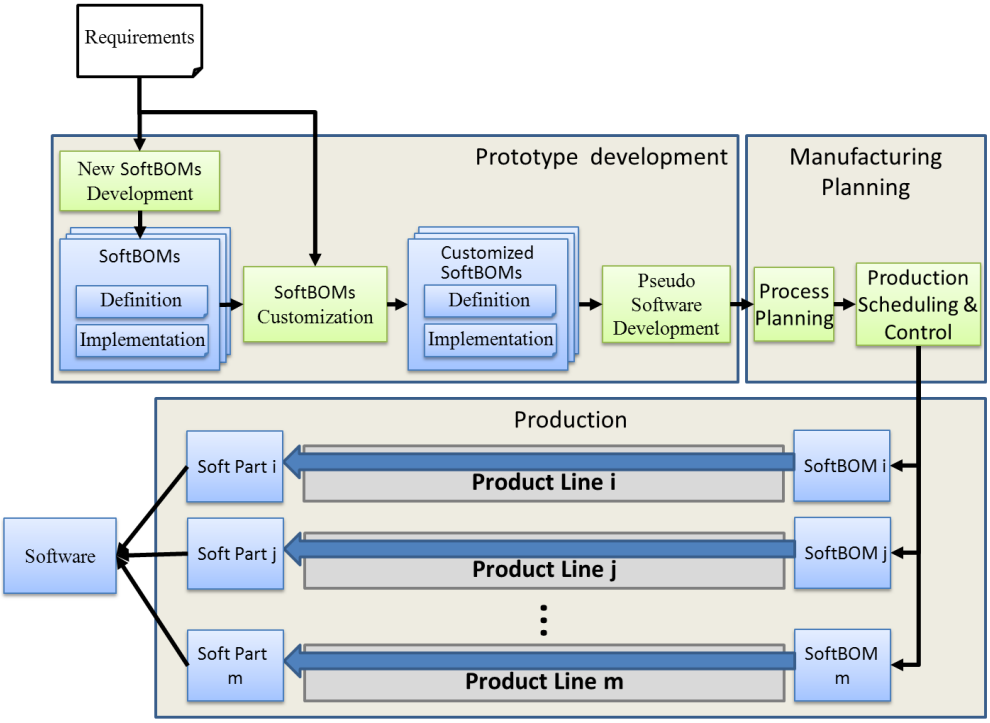

圖 1. 以虛擬軟體為基礎之軟體工廠生產模式

求的功能。首先，程式撰寫人員可重複使用 SoftBOM 範例程式中的程式碼來「組裝」出該 SoftBOM 的基本功能，再依 SoftBOM 虛擬軟體需求模型實作出需要客製的功能及可執行的測試案例程式，待所產生的軟體模組通過該 SoftBOM 的功能測試後，即進入下一個整合階段。當軟體工廠進入整合階段時，由於各別的生產線均已完成各個部份所必須達成的功能測試，我們只需在這個階段針對各生產線所產生出的成品進行整合測試，而後即發展出目標軟體。

4.2 軟體工廠達交預估系統之設計

一般而言，目前有關達交預估的做法，如 COCOMOII，是以數學模型作為預估基礎。利用數學模型預估的困難處，其中之一是在於無法全然將工廠本身的特性，正確而完整地建立到模型中。在軟體訂單達交的預估上，不同的工作派工勢必會影響排程結果。然而軟體工作派工牽扯因素頗多，並非單一數學模型所能預估。更有甚者，一個實際在運行的工廠，是一個動態變化的環境。因此，能將工廠本身的特質與經驗，例如像工廠本身資源的使用特性、限制條件、派工法則等，建立到預測模型中，再經由模擬來預估達交，才能夠更貼近軟體工廠在達交預估上的實際需要。同時，透過模擬機制，生產規劃系統所需的派工與生產排程建議也才能獲得。

圖 2 為達交預估系統之系統架構圖，同時，本計劃設計一組語法，以讓軟體專案團隊能依各軟體工廠的特性，來撰寫符合本身資源的使用特性、限制條件與派工法則，並經由本系統模擬產出該軟體工廠的派工與排程結果，進而完成預估達交及顧客交期回覆。預估達交系統同時也會產出符合預估達交的相關派工與排程資訊，進而作為軟體工廠生產規劃時的重要依據。

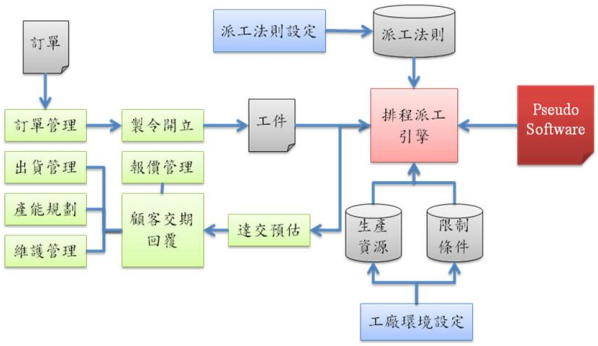

圖 2. 達交預估系統之系統架構

排程派工引擎為整個達交預估系統的核心部份，軟體開發團隊依各別團隊目前的狀況定義其派工法則，依待生產之工作、各職能之人力資源、工件內容及其他專案相關資訊等，以派工法則語法加以描述後，作為排程派工引擎主要參考依據及限制。排程派工引擎也依軟體工廠所能處理的專業領域的不同，依虛擬軟體中所描述的需求模型，動態調適每個工作項目的工時及完成複雜度，例如某需求之系統分析、設計及程式撰寫等。另一方面，每個軟體工廠所能開發的生產線模型依需求的變異，而可能有不同的環境設定需求，這將以軟體工廠中的生產資源與限制條件加以描述，我們將在下節詳細定義生產資源與限制條件的資料結構。最後，排程派工引擎將依據派工法則、虛擬軟體、工廠的生產資源與限制條件作為輸入條件，以簡單的數學模型推算出軟體的交期及專案管理相關資訊，並交由達交預估模組整合計算結果。本系統假設製令開立可由人工確定。接下來我們介紹生產資源與限制條件之設計：

- Project－為訂單名稱，包含以下欄位：
  - costUpperBound：此訂單客戶能接受之最高費用值。
  - expectedDueDate：此訂單客戶預期之交期。
- Work Item－待生產之工作，例如某需求之系統分析、某需求之系統設計、某需求之程式撰寫、或是某系統之測試等。由以下欄位所構成：
  - requiredSkill：所需之工作技能。
  - skillLevel：所需技能之成熟等級。
  - estimationTime：預估完成工件之標準工時。
  - priority：工件優先順序。
- Production People－為軟體工廠中各職能之人力資源，由以下欄位所構成：
  - id：人員代號。
  - skillSet：該人員所具備之技能集合。
  - cost：該人員工時成本。
- performanceOfSkill(skill)：該人員在技能 skill 上之產出基本效能。當此數值大於 1 時，代表該人員產值表現高於標準工時；反之，則低於標準工時。
- CalendarOf (x, date)－生產人員 x 在日期 date 的剩於可用工時比例。當此值為 0 時，代表該員可能休假或已被全部派工。此值最大值為 1，最小值為 0。
- SetOfPP－軟體工廠中，所有生產人員的總集合。
- QueueOfWI(x)－專案 x 中，尚未派工與完工的工件集合。
- PrerequisiteOfWI(x)－工件 x 需要前置完成的工件集合。
- TeamOfPP(x)－預計可供專案 x 使用的生產人員集合，為 SetOfPP 的子集合。
- ConstrainSetOfPW(x)－專案 x 中,預定人員與工件的相依關係集合，集合中之元素以結構[x, wi]表達之，代表工件 wi 特別指定需由人員 x 負責完成。
- ConstrainSetOfPS(x)－專案 x 中,預定人員與在此專案中所能使用技能之相依關係集合，集合中之元素以結構[x, skill]表達之，代表人員 x 在此專案中，僅能使用技能 skill。
- CurrentProgress(x)－紀錄專案 x 執行至目前的相關資訊。
- totalCost：至目前所有成本，計算規則為所有人員實際花費工時乘以各人員工時成本的總合。

上述生產資源與限制條件之設計，可以用來真實描述一個軟體工廠針對某訂單的生產資源與限制條件。例如，一個軟體工廠中，雖然有某工作人員既可以設計系統，但也可以撰寫程式。此時，針對某特定訂單專案，假設軟體工廠已認知某項工件非此人員撰寫不可，在本研究所發展的達交預估系統下，便可以明確建立此限制條件。

圖 3 為根據上述生產資源與限制條件所設計的派工法則之描述語法。在此語法中，軟體工廠可以實際描述其所需之派工法則。

```
<assign>::= <var> = <const>;
<build>::=Build <set> from <set>;
<build>::=Build <set> from <set> and <set>;
<build>::=Build <set> for all <var> in <set>;
<select>::=Select any <var> from <set> and
          select any <var> from <set> where <rule>;
<set>::=QueueOfWI|SetOfWI|TeamOfPP|SetOfPP|
        ConstrainSetOfPS |PrerequisiteOfWI|
        ConstrainSetOfPW
<rule>::=<rule> or <rule>,
<rule>::= expression,
<var>::= string
<const>::=string
```

圖 3. 派工語法之設計

4.3 持續發展軟體工廠所需相關工具

我們希望能透過虛擬軟體來發展及驗證 Android 應用程式的需求模型，使虛擬軟體能夠具象化，令使用者能夠透過操作虛擬軟體來發展及驗證需求。我們針對 Android 應用程式的需求發展工具並實現虛擬軟體概念性框架，將需求以虛擬軟體模型保存。透過虛擬軟體需求模型能將需求具象化以循序的方式播放需求讓關係人能像真實軟體般地操作，也提供需求導覽地圖呈現需求靜態結構讓開發團隊掌握需求全貌。透過本工具讓需求擁有可操作、可閱讀以及完整之特性，以減少溝通上的鴻溝並有效萃取出需求以滿足多變的手持裝置應用程式需求。

圖 4 為虛擬軟體發展工具的領域模型圖，Manager 負責管理虛擬軟體，虛擬軟體 的 Presentation 由 Widget 構成。Manager 藉著修改 Widget 之屬性以更改 Presentation；也能描述 Widget 的 Field Constraints。當場景繪製完成後，可以在場景與場景間建立 Navigation，由場景中 Widget 的 Event 所觸發，會導向另一個場景。然而場景轉換中可能有一些邏輯處理與資料傳遞是外觀所無法呈現的，因此可以為 Navigation 撰寫 Business Logic 以描述這些邏輯等。最後，撰寫 Test Cases 描述如何驗證 Business Logic 與 Field Constraint，作為開發時的參照。

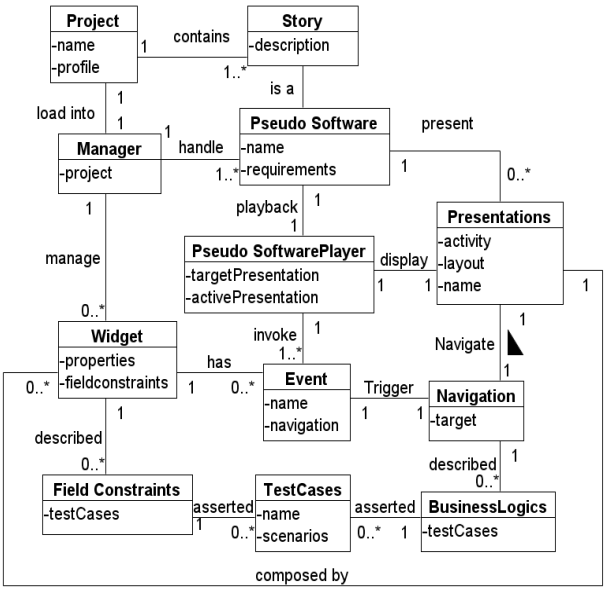

圖 4. 虛擬軟體發展工具之領域模型圖

五、結果與討論

5.1 達交預估系統之模擬派工法則

透過本計劃所提出之派工語法中，我們進一步用來描述符合軟體工廠實際狀況所需之派工法則。我們以模擬隨機派工法則與最適派工法則來說明，圖 5 所描述的是一個完全隨機的派工法則，只要人員能夠符合該工件之生產資訊與限制條件，即可派工；圖 6 則是根據目前最適合該工件的人員作為派工法則，所謂最適合的人員在此是指可以在最短時間完成該工件的人員。

```
p = Project;
Build QueueOfWI(p) from SetOfWI(p);
Build TeamOfPP(p) from SetOfPP;
Build ConstrainSetOfPS(p) from TeamOfPP(p);
Build PrerequisiteOfWI(x) for all x in QueueOfWI(p);
Build constrainSetOfPW(p) from QueueOfWI(p)
                                and TeamOfPP(p);
Select any wi from queueOfWI(p) and Select any x
    from TeamOfPP(p) where
       PrerequisiteOfWI(wi) is empty,
       ([wi, x] is in ConstrainSetOfPW(p)) or
       ([wi, _] is not in ConstrainSetOfPW(p)),
       (wi.requiredSkill is in x.skillSet),
       ([x, wi.requiredSkill] is in ConstrainSetOfPS(p)),
       CalendarOf(x,_) > 0.5,
       CurrentProgress.cost < p.costUpperBound;
```

圖 5. 模擬隨機派工法則範例

```
p = Project;
Build QueueOfWI(p) from SetOfWI(p);
Build TeamOfPP(p) from SetOfPP;
Build ConstrainSetOfPS(p) from TeamOfPP(p);
Build PrerequisiteOfWI(x) for all x in QueueOfWI(p);
Build constrainSetOfPW(p) from QueueOfWI(p)
                              and TeamOfPP(p);
Select any wi from queueOfWI(p) and Select any x
      from TeamOfPP(p) where
        PrerequisiteOfWI(wi) is empty,
        ([wi, x] is in ConstrainSetOfPW(p)) or
        ([wi, _] is not in ConstrainSetOfPW(p)),
        (wi.requiredSkill is in x.skillSet),
        ([x, wi.requiredSkill] is in ConstrainSetOfPS(p)),
        minimum(wi.estimationTime / CalendarOf(x,_)
        / x.PerformanceOFSkill(wi.requiredSkill));
```

圖 6. 模擬最適派工法則範例

5.2 軟體工廠所需相關工具 – PS4Android

在本年度的計劃中，我們持續發展支援虛擬軟體之需求發展工具，作為軟體工廠中用以定義欲開發軟體之需求模型，並加強使用者介面及功能以讓塑模人員能夠輕易操作。圖 7 為 PS4Android 中建立與設計虛擬軟體需求模型的畫面，主要分成二個區塊：①為專案工作區，根目錄為專案名稱；專案的子節點為情境的名稱；接著是情境中的每個步驟；每個步驟中則包含數個虛擬軟體的元素，其中 P 表示 Presentation；N 表示 Navigation；F 表示 Field Constraint；B 表示 Business Logic；T 表示 Test Case。②為虛擬軟體編輯區，讓塑模人員能夠進一步定義虛擬軟體各個元性的細節。

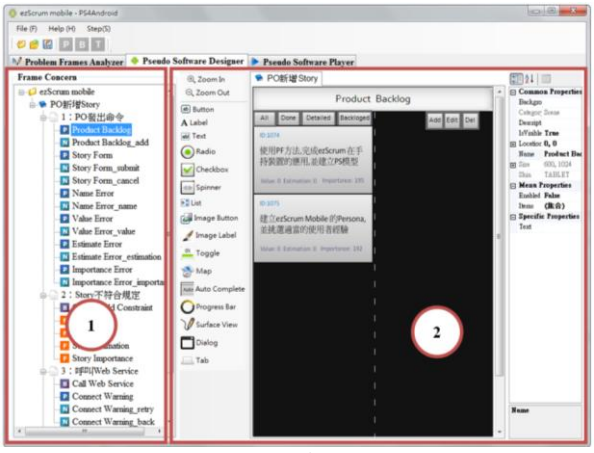

圖 7. 虛擬軟體需求模型設計畫面

為了能夠及時驗證虛擬軟體需求模型的正確性，在塑模人員在建立好虛擬軟體需求模型後，便能夠立即藉由模擬工具來驗證需求。圖 8 即為虛擬軟體之模擬畫面主要可分成三個區塊：

①為專案工作區，根目錄為專案名稱；專案的子節點為情境名稱；接著是情境中的每個步驟。使用者可以選擇一個情境後，點選 進行播放。

②為具有 Android 平板電腦外觀的播放畫面，在平板電腦的螢幕會呈現虛擬軟體中的 Presentation。

③呈現透過自然語言描述的 Business Logic 與 Field Constraint 內容。

在畫面中具有 Navigation 或是 Field Constraint 的元件會以邊框標示。當使用者按下具有 Navigation 的元件後，會根據使用者在 Pseudo Software Designer 中所定義的目的地而轉換至不同的 Presentation。如果此元件的目的地有多個選擇時，會出現選單提示使用者選擇一個，並在③顯示 Business Logic 的內容。若使用者點選的是具有 Field Constraint 的元件，則會在③顯示 Field Constraint 的內容。

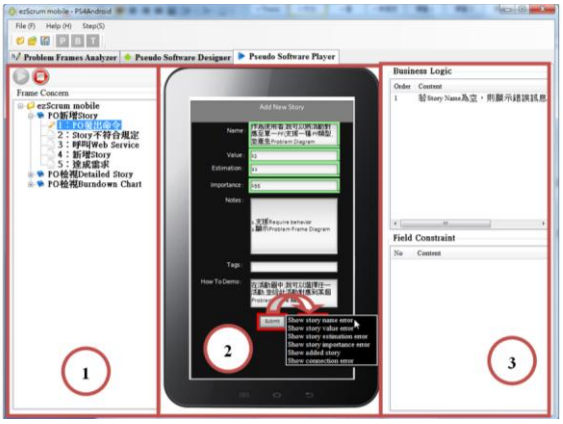

圖 8. 虛擬軟體需求模型模擬畫面

六、文獻探討

軟體工廠的概念早在 70 年代就被提出[12]，最早由 GE 的 Bemer 所提倡之標準化工具以提升生產率的方式[5]。而後其軟體工廠概念由日本企業應用[10]，並逐一在各地設立軟體工廠，如 Toshiba[26]、Hitachi[15]等。在 1980 年代於歐洲等地也

有 Aaen 提出的 Eureka Software Factory (ESF)[14,15,31]。90 年代則有 HP 的 M. L. Griss 所提出的特定應用領域、System Reuse 為主的軟體工廠[18]。而近年來隨著 MDA 興起，亦即有以 Model-Driven 類型的軟體工廠(Model-Based Software Factory)產生[17,22]; 另一方面也有強調以人為中心的軟體工廠類型(Cell-Based Software Factory)[43]。我們也參考了許多關於軟體生產線(Software Product Line)的相關研究[36, 37, 38, 40, 41, 42]，其生產一群功能相近但仍屬各自不同的軟體之軟體生產方式。

參考文獻

1. W.W. Agresti, “Software Engineering as Industrial Engineering,” Software Eng. Notes, vol. 6, no. 5, 1981, pp. 11-12.
2. V.R. Basili, G. Caldiera, and G. Canone, “A Reference Architecture for the Component Factory,” *ACM Transactions on Software Engineering and Methodology*, Vol. 1, Issue 1, 1992, pp. 53-80.
3. B. Blum, “Understanding the Software Paradox,” ACM SIGSOFT Software Engineering Notes, vol. 10, no. 1, 1985, pp. 43-47.
4. B. Boehm, *Software Cost Estimation with COCOMOII*, Prentice-Hall, 2000.
5. Bemer, R.W., Position papers for Panel Discussion: The Economics of Program Production, in A. J. H. Morrell (Ed.), Information Processing 68, Amsterdam, North-Holland (1968)
6. H. Bratman; and T. Court, "The Software Factory," *Computer*, vol.8, no.5, May 1975, pp. 28-37.
7. F.P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, 1975, pp. 47-48.
8. B. Clark, S.Devani-Chulani, and B. Boehm, “Calibrating the COCOMO II Post-Architecture Model,” *Proceedings of the International Conference on Software Engineering Workshop*, IEEE Computer Soc. Press, Los Alamitos, USA, 1998, pp.477–480.
9. J.O. Coplien, “Reevaluating the Architectural Metaphor: Toward Piecemeal Growth,” *IEEE Software*, vol. 16, no. 5, Sep./Oct. 1999, pp. 40-44.
10. M. A. Cusumano, *Japan's Software Factories : A Challenge to US Management*, Oxford University Press, 1991.
11. M. A. Cusumano, “Shifting Economies: From Craft Production to Flexible Systems and Software Factories,” *Research Policy*, vol. 21, 1992, pp. 453-480.
12. M. A. Cusumano, "The Software Factory: A Historical Interpretation," *IEEE Software*, vol.06, no.2, March/April, 1989, pp. 23-30.
13. M. W. Evans, *The Software Factory*, Wiley, New York, 1989.
14. C. Fernström, “The Eureka Software Factory: Concepts and Accomplishments,” *Proceedings of the 3rd European Software Engineering Conference*, LNCS No. 550, pringer-Verlag, 1991, pp. 23-36.
15. C. Fernstrom; K.-H. Narfelt; and L. Ohlsson, "Software Factory Principles, Architecture, and Experiments," *IEEE Software*, vol.9, no.2, pp.36-44, Mar 1992
16. A. Fuggetta, L. Lavazza, S. Morasca, S. Cinti, G., Oldano and E. Orazi, “Applying GQM in An Industrial Software Factory,” *ACM Trans. Softw. Eng. Methodol.* 7, 4 (Oct. 1998), pp. 411-448.
17. J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories, Assembling Applications with Patterns, Models, Framework, and Tools*, Wiley, 2004.
18. M. L. Griss, and K. D. Wentzel, “Hybrid Domain-Specific Kits for A Flexible Software Factory,” In *Proceedings of the 1994 ACM Symposium on Applied Computing*, Phoenix, Arizona, United States, March 06 - 08, 1994, pp. 47-52.
19. J.D. Herbsleb, and D. Moitra, “Global Software Development,” *IEEE Software*, Vol. 18, Issue 2, 2001, pp.16-20.
20. W. S. Humphrey, “Software and The Factory Paradigm,” *Software engineering Journal*, Vol.6, Issue 5, September, 1991, pp. 370-376.
21. IEEE, “IEEE standard for developing software life cycle processes”, *IEEE Std 1074 – 1995* , April, 1996.
22. B. Langlois, and D. Exertier, “MDSoFa, A Model-Driven Software Factory,” *OOPSLA* 2004, MDSD Workshop.
23. C. Li, H. Li, and M. Li, “A Software Factory Model Based on ISO 9000 and CMM for Chinese Small Organizations,” *Proceedings of the Second Asia-Pacific Conference on Quality Software*, December, 2001, pp.288–292.
24. N.K. Lim, J.S.K. Ang, and F.N. Pavri, “Diffusing Software-Based Innovation with A Software Factory Approach for Software Development,” *Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology*, Vol. 2, November, 2000, pp. 549 -555.
25. M. S. Mahoney, "Finding a History for Software Engineering," *IEEE Annals of the*

History of Computing, January–March 2004, pp. 8-19.
26. Y. Matsumoto, "Japanese Software Factory," in *Encyclopedia of Software Engineering*, (ed.) Marciniak, J.J., FIRST EDITION, Johon Wiley & Sons, N.Y, pp.593-605.
27. J. Nandhakumar, "Managing Time in a Software Factory: Temporal and Spatial Organization of IS Development Activities," *The Information Society*, Volume 18, Number 4, 1 July 2002 , pp. 251-262(12)
28. L.J. Osterweil, "Software Processes are Software Too," *Proc. 9th Int'l Conf. Software Eng. (ICSE 9)*, IEEE Computer Soc. Press, 1987, pp. 2-13.
29. K. Pillai, and V.S. S. Nair, "A Model for Software Development Effort and Cost Estimation", *IEEE Transactions on Software Engineering*, Volume 23, Issue 8, August, 1997 pp: 485 – 49.
30. K. Potosnak, "Human Factors-Management: The Key to Success", *IEEE Software*, Volume 6, Issue 2, March, 1989, pp: 86 – 88.
31. W. Schäfer, and H. Weber, „European Software Factory Plan—the ESF profile," In *Modern Software Engineering, Foundations and Current Perspectives*, P. A. Ng and R. T. Yeh, Eds. Van Nostrand Reinhold Co., New York, NY, 1989, pp. 613-637.
32. Standish Group, "Extreme CHAOS (2001)," 2001, *http://www.standishgroup.com/sample_research/*
33. K. Swanson, D. McComb, J. Smith, and D. McCubbrey, **"**The Application Software Factory: Applying Total Quality Techniques to Systems Development,", *MIS Quarterly*, Vol. 15, No. 4 (Dec., 1991), pp. 567-579.
34. H. Weber, *The Software Factory Challenge*, IOS Press, 1997.
35. C. Atkinson et al., Component-based product line engineering with UML. Addison-Wesley, London, New York, 2002.
36. P. Clements, L. Northrop, Software Product Lines: Practice and Patterns, Addison Wesley, Reading MA, 2001
37. P. Clements, and C. Krueger, Being Proactive Pays Off/Eliminating the Adoption Barrier, IEEE Software, Special Issue of Software Product Lines. July/August 2002, pages 28-31
38. H. Gomma, Designing Software Product Lines with UML, Addison-Wesley, 2004.
39. Jwo, J.-S., Cheng, Y.C., Pseudo software: A mediating instrument for modeling software requirements. J. Syst. Software (2009), doi:10.1016/j.jss.2009.10.042
40. Charles W. Krueger, Easing the Transition to Software Mass Customization, Proceedings of the 4th International Workshop on Product Family Engineering, October 2001, Bilbao, Spain, Springer-Verlag, New York.
41. Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Känsälä and Henk Obbink, Software Product Family Evaluation, in Proceedings of the 3rd International Software Product Line Conference, Boston, MA, USA, Aug. 30-Sep. 2, 2004.
42. D. Weiss, C. Lai, and R. Tau, Software product-line engineering: a family-based software development process, Addison-Wesley, Reading, MA, 1999.
43. Yoshihiro Matsumoto site, http://www5d.biglobe.ne.jp/~y-h-m/

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■ 達成目標

說明：

在本計劃中，我們發展軟體工廠所需之相關工具，使得類似於硬體生產的軟體工廠生產方式成為一種可行的方案。另一方面，本計劃也提出一個軟體生命週期管理的快速導入方法，用以幫助軟體團隊簡單且快速導入軟體生命週期管理工具。

研究成果在學術期刊發表或申請專利等情形：

論文：■已發表 □未發表之文稿 □撰寫中 □無

本計畫執行至今的研究成果已逐一發表在國內外的會議及期刊上：

國際期刊與研討會論文:

- 論文「Jumpstarting Application Lifecycle Management: a New Approach with Tool Support」已被國際期刊 Journal of Information Science and Engineering 接受
- 論文「Pseudo Software: a Mediating Instrument for Modeling Software Requirements」 已發表至國際期刊 Journal of Systems and Software, Volume 83 , Issue 4 (April 2010), Pages: 599-608
- 論文「Rapid Application Lifecycle Management: a new Approach with Tool Support」 已發表至國際研討會 SoMeT2010。

國內研討會論文:

- 論文「一個以 Pseudo Software 為基礎之 Android 需求發展工具」已發表至第五屆台灣軟體工程研討會論文集(TCSE2009)
- 論文「一個結合 Problem Frames 與 Pseudo Software 的需求發展方法」已發表至第二十一屆物件導向技術及應用暨第六屆台灣軟體工程研討會論文集

請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）（以 500 字為限）

計畫執行至今已獲致如下成果：

1. 提出 Pseudo Software 新概念：基於 Pseudo Software，將 IKIWISI (I’ll Know It When I See It)的困擾得以提前在需求階段解決外，更重要的是需求能夠以 Pattern 的方式記錄下來。同時 Pseudo Software 也可以與軟體工廠的 SoftBOM 存在相對應關係，如此才能夠使軟體工廠能夠模仿硬體工廠的生產模式。
2. 提出一個軟體生命週期管理的快速導入方法，稱為 RALM。RALM 首先提供一個軟體生命週期管理的參考模型與定義樣版。透過參考模型與定義樣版，中小型軟體團隊可以快速發展出所需的軟體生命週期管理定義。而發展完成的軟體生命週期管理需求，可進一步利用 RALM 所提供的工具 ALMTranslator 產出 VSTS 流程範本以及 VSTS 設定指引，幫助軟體團隊簡單且快速導入微軟公司的 VSTS 軟體生命週期管理工具。

附件四

國科會補助專題研究計畫項下出席國際學術會議心得報告

日期: 99年10月21日

計畫編號	NSC 98－2221－E－029－017－		
計畫名稱	適用於企業計算之軟體工廠模式-III		
出國人員姓名	周忠信	服務機構及職稱	東海大學資訊工程學系
會議時間	99年9月28日 至 99年10月1日	會議地點	日本橫濱
會議名稱	(中文)第九屆軟體方法、工具與技術國際研討會 (英文) The 9th International Conference on Software Methodologies, Tools and Techniques (SOMET_10)		
發表論文題目	(中文)快速應用生命週期管理：方法論與支援工具 (英文)Rapid Application Lifecycle Management: a new Approach with Tool Support		

一、參加會議經過

The 9th International Conference on Software Methodologies, Tools and Techniques(SOMET_10) 於 9/29~10/1 在日本橫濱召開。此次會議由岩手縣立大學、日本學術振興會、SANGIKYO、Microsoft 等共同主辦，與會者主要來自人工智慧與軟體工程領域的專家學者。我與鄭有進教授等合著的論文 Rapid Application Lifecycle Management: a new Approach with Tool Support 被安排於 9/30 日宣讀。本次的論文發表由我來向其他專家學者說明研究成果，現場反應良好，也和其他學

者交換意見及研究上的想法。特別是在晚宴期間與 Brian HENDERSON-SELLERS 就本次發表的研究成果進行討論與意見交換(HENDERSON-SELLERS 為國際軟工學界著名的學者，發表過許多軟體工程的方法)。HENDERSON-SELLERS 認為 RALM 的概念與幾年前他所提出的方法概念上相近，但能夠實現在近年來的主流 ALM 工具上，如 Microsoft VSTS 上，使得研究成果具有打進主流工具的潛力。

(上圖即為 9/30 當日宣讀研究成果的照片)

二、與會心得

近年來，由於行動運算與雲端運算已成為主流趨勢之一，不論學界、產業界均能感受到巨大的產業變革，像是 APP 與雲端運算服務的興起，均對目前的產業及研究上產生更多值得討探的議題。然而，在這波變革中，軟體工程(包含軟體技術、軟體設計、軟體流程、軟體工具等)扮演著極為重要的重要領域之一。而雲端運算服務

除了一般個人應用外，小型或微型企業願意採用雲端運算服務，會是雲端運算服務成功的關鍵因素。所謂雲端運算服務，簡單說是指將軟體當成服務。感覺上似乎很簡單，因為將軟體放在網路上，無論是透過虛擬化(virtualization)技術，或是將之重構成為網路應用軟體(web applications)，基本上都不是一件很困難的工作。然而，要將軟體變成以租賃為導向的雲端運算服務，涉及的技術挑戰將變得很高，其中包括支持多租戶管理相關的「個別租戶的安全獨立性」(security isolation)、「個別租戶的效能獨立性」(performance isolation)、「個別租戶的可用獨立性」(availability isolation)、「個別租戶的管理獨立性」(administration isolation)、以及「個別租戶的軟體需求客製」(customization)等。這些問題的解決，將有賴於軟體工程在發展軟體上，現有的軟體技術、軟體設計、軟體流程、軟體工具能否支援這類雲端運算服務的開發。

本次會議有看到許多相當有趣的論文，其中最主要的一篇為 9/29 的 invited talk："Consolidating diagram types from several agent-oriented methodologies"，講者為 Brian HENDERSON-SELLERS (Faculty of Information Technology, University of Technology, Sydney, Australia)，作者利用 semantics net、agent、ontology 等 AI 技術，描述、分析、比較與改善 agent-oriented software development 的方法。這種運用 AI 技術來改善軟體工程的方法，可作為未來研究的參考方向。

三、考察參觀活動(無是項活動者略)

無。

四、建議

國內學界每年均有相關領域學會舉辦會議，過去，仍少見以二個以上學會聯合辦理者。因此，關於軟工領域中使應其他領域的方法、或其他領域使用軟工的方法的研究論文，通常無法在兩個領域學者同時在場的場合下進行宣讀，造成知識交流上的缺憾。今年 7 月，軟工學會與物件導向 SIG 第一次將其年會合併辦理(由中大與北科大共同主辦)，以期擴大交流與參與，可視看為一個開端。往後 貴會資助舉辦研討會時，不妨鼓勵跨領域的二個（或以上）學會一起辦理會議。

五、攜回資料名稱及內容

研討會論文集專書一本：『New Trends in Software Methodologies, Tools, and Techniques,』 Hamido Fujita ed. IOS Press，2010. ISBN 978-1-60750-628-7

六、其他

無。

Rapid Application Lifecycle Management: a new Approach with Tool Support

Jung-Sing Jwo[a], Yu Chin Cheng[b], Tien-Song Hsu[b], Chun Hsin Liu[a]

[a] *Department of Computer Science, Tunghai University, Taichung, Taiwan*

[b] *Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan*

Abstract. Software lifecycle is the process by which software is conceived, developed, maintained, and decommissioned. To the development team, initiating effective application lifecycle management (ALM) is challenging for three reasons. (1) ALM definition is hard since lifecycle activities are interdependent and complex in nature that involves product, project, people, process, tool and technology. (2) ALM activities require the support of correctly tailored tools. (3) Effective ALM activities execution requires discipline. To take on the three challenges of initiating ALM, we present a new approach called Rapid Application Lifecycle Management (RALM). RALM provides a reference model with a number of templates for ALM activity definition. Once customized, the templates are converted into platform-specific process definition files with tool support. Observations from a field application of RALM are presented and discussed .

Keywords. Application lifecycle management, ALM definition, ALM discipline, ALM implementation, VSTS

1. Introduction

Application lifecycle is the process by which an application is conceived, developed, maintained, and decommissioned [12, 14]. The "health" of the application very much depends on the lifecycle activities that take place. For instance, inadequately defined requirements defy the best intention to develop good software; insufficiently defined issue tracking states can prevent the real cause behind high defect rates from being exposed; and so on. Typical lifecycle activities include requirements development, project planning, solution development, requirements management, deployment, issue tracking, and so on [1, 2, 4]. Each of these lifecycle activities can involve a multitude of technologies and tools. In this paper, we assume that the development processes and activities are defined, executed, adapted, and managed on state-of-the-art application lifecycle management (ALM) platforms, which are described in Section 2. While the use of ALM platforms reduces the complexity of integrating development and management tools and allows the team to quickly modify process as required, to the development team, the first hurdle lies in jumpstarting effective application lifecycle management for a software development project on the target platform. Initiating ALM is challenging in three respects:

- *ALM definition*: A formal ALM definition helps software engineering team implementing the ALM on the selected ALM platform. However, for most of the small-to-medium software teams, formally defining the ALM activities and their interrelationships is in itself a very complicated undertaking [3, 18].
- *ALM discipline*: ALM activities remain difficult to implement even after they are defined. In particular, coordination and cooperation among developers are often viewed as non-technical overhead and there is usually a lack of willingness among the team members to comply [17].
- *ALM implementation*: ALM activities require tool support for their implementation. Commercial and open source ALM tools require tailoring to suite the team's need. Again, misfits are common: a tool can offer too many un-wanted features while some critical features may be missing [11].

To date, several approaches have been reported in the ALM literature. In order to avoid misfits brought by adopting an ALM platform, Moore et al. suggest making modifications to the standard templates or even

creating new templates from scratch [11]. Kääriäinen J. et al. have defined an ALM framework for documenting the company's ALM solutions and finding improvement for ALM solutions [9]. Pirklbauer G. et al. identify key problem areas typically addressed by ALM and provide guidance on how to develop an ALM strategy for software development organizations [13]. In this paper, we present an instrumented approach for ALM definition, ALM discipline and ALM implementation. The approach is called *Rapid Application Lifecycle Management* (RALM). RALM provides a reference model for ALM that can be used by software team to jumpstart the task of ALM definition. The reference model is defined in a number of templates which are in Microsoft Excel format. The conversion from templates to platform-specific settings is done by a tool named *ALMTranslator*. The current version of ALMTranslator can be used to implement the defined ALM activities on the Microsoft Visual Studio Team System (VSTS) 2005. In addition to the ALM implementation, engineering guidelines for carrying through *ALM discipline* are also automatically published on the project portal by ALMTranslator.

2. State-of-the-art ALM platforms

The development of ALM platforms and tools (e.g., Microsoft VSTS [16], Borland ALM [5], and IBM Jazz [8]) has reached a point that they emphasize the customization of processes and tools to suit the development team's need. No longer are teams forced to accept a specific process, especially one that the team is unfamiliar with. As a result, the same ALM platform can support waterfall, unified, agile, or any other home grown software processes [15]. The main features of ALM platforms include configuration and change management, issue tracking, project management, build management, report management, and so on. As can be seen, substantial amount of work is involved in setting the platform for these functions to work correctly.
State-of-the-art ALM platforms fill the "how-to" gap for organizations that adopt process improvement standards such as CMMI [6], which provides a framework for defining "what" to be done. For example, in configuration management, a support process area in CMMI, a specific practice says that access control should be established. However, until the project has been created and configuration items have been determined, such a practice is at best a declaration; access control setting is only possible after project artifacts have been defined. For this reason, it can be noted that while an ALM platform such as VSTS supports CMMI, customizations such as setting access control rights are still necessary and are performed manually [16].

3. The RALM Process

The primary benefit of a comprehensive ALM platform is the provision of all commonly used ALM functionalities (or best practices) on a single platform. However, the numerous interdependent ALM activities also greatly increase the complexity and difficulty of adopting the ALM platform. To avoid being overly general, we propose to look at ALM at the project level. That is, once the project scope, budget, and time are known, many activities can be defined down to the specifics, which help eliminating vagueness and avoiding misfit. The RALM process is depicted in Figure 1, where VSTS is used as the target ALM platform although the process is equally applicable to other ALM platforms including other commercial and open source ones. RALM process consists of two phases: *ALM definition* and *ALM platform initiation*.

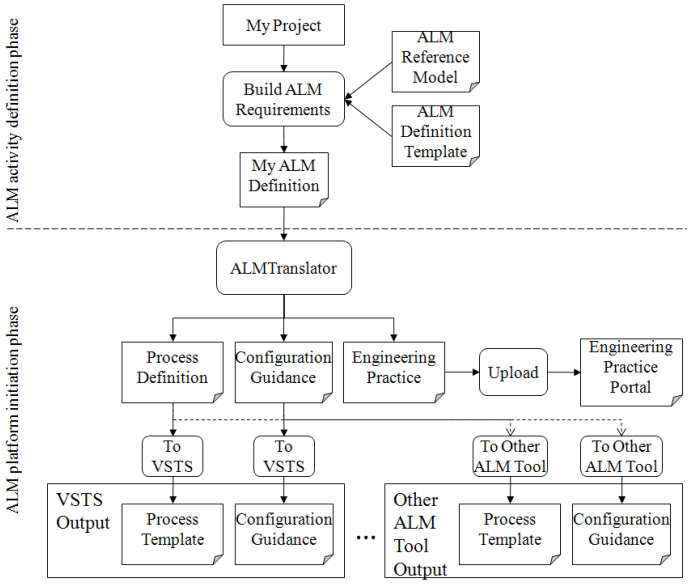

Figure 1. RALM process

In the ALM definition phase, the development team picks a project ("My Project" in Figure 1). An ALM reference model is provided to help the development team capture and define its ALM activities. The activities inside the reference model are currently known best practices, including requirements management, change management, issue tracking, configuration management, project management, and so on. First, the current practices of the project team are selected and sequenced into a development scenario, which is reviewed against the reference model. Each ALM activity is captured in an activity template. The template has an advantage in allowing the captured ALM activity definition to be reviewed in a time efficient manner. Alternatively, on an ALM platform like VSTS, the definition can be directly configured into the platform. By so doing, however, the settings become invisible: to review the settings the team must actually play with the ALM platform, which can be a cumbersome and time-consuming process. Also, verifying satisfaction becomes difficult since only a partial view of the ALM definition is revealed. In contrast, with the use of templates the task of ALM activity definition is iterated until the team comes to an agreement. The end of the definition phase produces a collection of customized ALM activities, which is collectively called "MyALM definition" in Figure 1.

In the ALM platform initiation phase, the ALM activity definitions are implemented on the target ALM platform. Done manually, the task of setting the ALM platform according to the definition tends to be tedious and error-prone. Fortunately, state-of-the-art ALM platforms allow batched implementation of ALM definitions through what are called the *process definition files*. To take advantage of batched setting, RALM converts the customized ALM activity definitions into process definition files through a conversion tool (ALMTranslator in Figure 1), which generates three types of files: the process definition files to be imported to the target ALM platform, a configuration guide for the target ALM platform, and engineering practice guidance which is published on the project portal. In our example, the process definition files are imported into VSTS. Since not all settings can be accomplished through imports, the remaining settings (e.g., the access control rights for team members) are performed manually according to the configuration guidance.

4. The design of ALMTranslator

Figure 2 shows the design class diagram of ALMTranslator, which consists of ALMParser, ALM2VSTS and PracticeGenerator. The ALM definitions files are parsed into the ALM-XML model by the ALMParser. The ALM-XML model is manipulated by the ALM2VSTS for generating the platform-specific process definition files and the configuration guidance. ALM2VSTS also needs to get the configurations of target platform through the specific ConfigLoader (VSTSConfigLoader in this case). Finally, the PracticeGenerator generates the platform-independent engineering practice from the ALM-XML model.

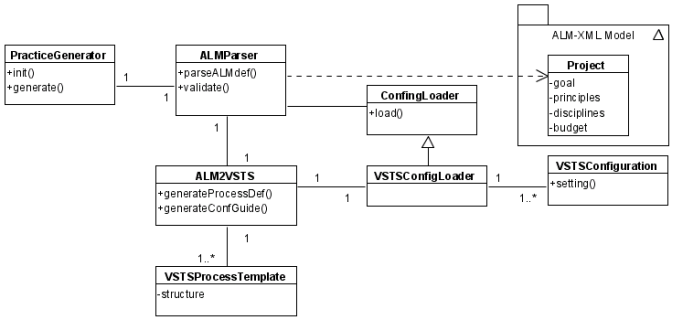

Figure 2. The design class diagram of ALMTranslator

5. Observations from field application and discussion

RALM has been applied to jumpstart the development of a balanced score card (BSC) application in an enterprise solution provider. The waterfall process was adopted. The project duration was three months and involved 11 developers from both headquarter and an offshore location. Each developer has a different access control right to the code repository. The engineering practices adopted included configuration management, issue tracking. The reports included bug rates, remaining works, and burn-down charts. Due to space limit we will not go into detail. By applying RALM, it was observed that

- the project initiation phase, which included the ALM definition from the RALM templates, VSTS platform instrumentation, and training, was completed in a work week; and
- the configuration guidance generated by RALM's ALMTranslator enhanced the built-in guidance provided by VSTS, for example, in proving a step-by-step guide for setting the issue types and states in issue tracking and in setting the access control rights to the team members.

According to the classification of ALM tools by Goth [7], RALM can be seen as a combination of *consulting component* and *ALM plug-in*. Since each team has its own needs, the challenge is to find suitable implementations of ALM for complicated, real-life situations. This is has been reported even for teams that belong to the same organization, where several case studies involved using VSTS as the platform for running Scrum [11]. Medina-Domínguez et al. proposed the adaption of process template based on the project patterns and a model to support process improvement by using patterns in a TFS environment [10]. In these works, the tailoring of the ALM platform was performed manually. RALM can obviously help by proving a more streamlined initiation.

6. Conclusion

RALM, a new method for jumpstarting ALM practices on state-of-the-art platform such as VSTS, has been proposed. RALM is facilitated with tool support, e.g., ALMTranslator to convert the ALM definitions into process definition files for VSTS. With ALMTranslator, process specialists adopting the RALM method will define, review, and revise ALM process in a spreadsheet. Once validated, the definition is converted into process definition files in VSTS. This not only expedites the ALM initiation, but also makes the definition visible to all. With respect to the three challenges outlined in the Introduction, RALM achieves the following benefits:

- The ALM reference model and templates provide a framework to facilitate the organization to define its own ALM activities. In the organization, this helps teams adopt their own ALM activities and reduce the complexity of initiating ALM activities.
- The software engineering guidelines is published on the project portal. The availability of such information helps the team in abiding to the disciplines.
- The process of initiating ALM tool support is expedited and the problem of ALM platform misfit is effectively resolved.

7. Acknowledgement

This research is funded by the National Science Council of Taiwan under grant contracts 98-2221-E-029-022 and 98-2221-E-027-049-MY3, and a grant from MOEA under grant contract 98-EC-17-A-02-S1-135.

References

[1] Alfonso Fuggetta, “Software process: a roadmap”, Proceedings of the Conference on The Future of Software Engineering ICSE '00, May, 2000, pp:25 – 34.

[2] Alvin W. Yeo, “Global-software development lifecycle: an exploratory study”, Proceedings of the SIGCHI conference on Human factors in computing systems, March, 2001.

[3] Bennett, K. H., and Rajlich, V. T., “Software maintenance and evolution: a roadmap,” Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 73 – 87.

[4] Bill Curtis, Marc I. Kellner, Jim Over, “Process modeling”, Communications of the ACM, Volume 35, Issue 9, September, 1992, pp:75 – 90.

[5] Borland ALM, http://www.borland.com/us/solutions/index.html.

[6] Chrissis, M. B., Konrad, M., Shrum, S., CMMI: Guidelines for Process Integration and Product Improvement, Addison-Wesley, 2003.

[7] Goth, G., “Agile Tool Market Growing with the Philosophy,” IEEE Software 26(2), pp. 88–91, 2009.

[8] IBM Jazz, http://jazz.net/.

[9] Kääriäinen, J., Välimäki, A., "Applying application lifecycle management for the development of complex systems: Experiences from the automation industry," Communications in Computer and Information Science 42, pp. 149-160, 2009

[10] Medina-Domínguez, F., Sanchez-Segura, M., Amescua, A., García, J., “Extending Microsoft Team Foundation Server Architecture to Support Collaborative Product Patterns,” In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 1–11. Springer, Heidelberg, 2007

[11] Moore, R., Reff, K., Graham, J., Hackerson, B., “Scrum at a Fortune 500 Manufacturing Company,” In AGILE 2007, pp. 175–180, 2007

[12] Pfleeger, S. L., “The nature of system change”, IEEE Software, Vol. 15, Iss. 3, June, 1998, pp. 87-90.

[13] Pirklbauer, G., Ramler, R., Zeilinger, R., “An integration-oriented model for application lifecycle management,” ICEIS 2009-11th International Conference on Enterprise Information Systems, Proceedings ISAS, pp. 399-402, 2009

[14] Rajlich, V.T., and Bennett, K.H., “A staged model for the software life cycle,” IEEE Computer, Vol. 33, Iss. 7, July, 2000, pp. 66-71.

[15] Sam Guckenheimer, Juan J. Perez, “Software Engineering with Microsoft Visual Studio Team System”, Addison Wesley, 2006.

[16] Semeniuk, J. and Danner, M., Managing Projects with Microsoft Visual Studio Team System, Microsoft Press, 2007.

[17] Shaw, M., “Prospects for an engineering discipline of software,” IEEE Software, Vol. 7, Iss. 6, November 1990, pp. 15-24.

[18] Turpin, R., “A progressive software development lifecycle”, Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems, October, 1996, pp. 208-211.

國科會補助計畫衍生研發成果推廣資料表

日期:2011/10/25

國科會補助計畫	計畫名稱: 適用於企業計算之軟體工廠模式(III)
	計畫主持人: 周忠信
	計畫編號: 99-2221-E-029-017- 學門領域: 程式語言與軟體工程

無研發成果推廣資料

99 年度專題研究計畫研究成果彙整表

計畫主持人：周忠信	計畫編號：99-2221-E-029-017-
計畫名稱：適用於企業計算之軟體工廠模式(III)	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	2	2	100%		
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	3	3	100%	人次	
		博士生	1	1	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
國外	論文著作	期刊論文	2	2	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	1	1	100%		
		專書	0	0	100%	章/本	
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		

其他成果 (無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)	無

	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■達成目標

□未達成目標（請說明，以 100 字為限）

□實驗失敗

□因故實驗中斷

□其他原因

說明：

2. 研究成果在學術期刊發表或申請專利等情形：

論文：■已發表 □未發表之文稿 □撰寫中 □無

專利：□已獲得 □申請中 ■無

技轉：□已技轉 □洽談中 ■無

其他：(以 100 字為限)

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）(以 500 字為限)

計畫執行至今已獲致如下成果：

1.提出 Pseudo Software 新概念：基於 Pseudo Software，將 IKIWISI (I’ ll Know It When I See It)的困擾得以提前在需求階段解決外，更重要的是需求能夠以 Pattern 的方式記錄下來。同時 Pseudo Software 也可以與軟體工廠的 SoftBOM 存在相對應關係，如此才能夠使軟體工廠能夠模仿硬體工廠的生產模式。

2.提出一個軟體生命週期管理的快速導入方法，稱為 RALM。RALM 首先提供一個軟體生命週期管理的參考模型與定義樣版。透過參考模型與定義樣版，中小型軟體團隊可以快速發展出所需的軟體生命週期管理定義。而發展完成的軟體生命週期管理需求，可進一步利用 RALM 所提供的工具 ALMTranslator 產出 VSTS 流程範本以及 VSTS 設定指引，幫助軟體團隊簡單且快速導入微軟公司的 VSTS 軟體生命週期管理工具。