Extreme Programming 與 CMM/CMMI 於小型開發團隊適用性之探討與比較

廖峻鋒
政治大學資訊科學所
智慧型媒體實驗室
try@nccu.edu.tw

摘要

XP(eXreme Programming) 是由 Kent Beck 於 1996 年代初期針對小型團隊提出的軟體工程方法論，可以同時滿足開發人員的成就感與提高軟體品質。在小型專案的應用上，XP 相當成功。另一個最近受到重視的軟體工程方法論是 CMM/CMMI，它是 CMU 的軟體工程學院在 1986 年開始發展的一套軟體工程能力成熟度模型，研究指出 CMM 亦適用於小型組職上[1]。大學院校研究單位及資訊科系實驗室的開發團隊通常以小型團隊為主，而軟體公司生產開發部門也大多由小型組織組成。這篇報告的目的在於分析這二個不同的方法論的特色，並討論如何才能將他們應用在小型團隊上。本文同時探討了一個這類型團隊的案例，可以做為小型軟體開發團隊建立開發程序時的參考。

1. 導論

XP(eXreme Programming) 是由 Kent Beck 於 1996 年代初期提出。XP 從人的本質考慮如何讓程式設計師，可以有尊嚴的工作，同時可以產出高品質的軟體。在軟體開發的過程中 XP 提出 4 個核心價值觀及 12 項的基本實務作法。XP 注重單元測試、雙人組程式設計及程式重構(refactoring)的工作，被認為是適合用在小型專案及團隊的輕量級軟體工程方法論。

CMM/CMMI 是美國卡內基美隆大學的軟體工程學院(SEI)接受美國國防部委託、在 1986 年開始發展的一套軟體工程能力成熟度模型。原本 CMM/CMMI 是用來評鑑組織、部門或團隊進行軟體開發專案的能力水準的模型，此模型對專案從開發到完成的流程提供一個嚴謹完備的架構與描述，使得歷年來引進這套評鑑制度的企業享受到軟體開發時程明顯縮短、成本大幅降低、產品瑕疵數減少以及品質提升等等的好處。目前 CMM/CMMI 在美國、印度及中國大陸已經廣泛地被軟體開發商採用。

雖然 XP 與 CMM/CMMI 相比，在開發程序上較不嚴謹，很多論點也受到批評及爭議，但也有學者指出，XP 倡導了許多良好的軟體工程原則，若在合適的環境裏運用得當，能達成許多 CMM 第二級和第三級的目標[1]。從另一個角度來看，很多支持 XP 的人認為，小型團隊由於人手及時間不足，CMM/CMMI 許多嚴格的規範反而會對整個團隊增加太多負擔。針對這種說法，也有學者提出應用在 CMM/CMMI 在小型團隊中的解決方案[3]。

台灣大部份軟體公司的研發部門及各大學的實驗室都是都以小型團隊的方式運作、開發軟體。大型的研發團隊有較多的人力，較細的分工及較充足的資源，合適於大團隊的方法不見得能夠適用於小型團隊。因此我們會把焦點放在找出這二種方法論對於小型團隊的正面及負面影響，並建議一個適用於這類型團隊的可能應用方式。

2. CMM/CMMI 軟體成熟度模型

軟體發展能力的成熟度模型(Capability Manurity Model for Software，CMM)是軟體工程協會 SEI(Software Engineering Institution)在 CMU 大學開發完成，目的是針對一個組織的軟體發展能力進行評價所設的標準，CMM 重於對軟體發展過程和開發方法論的考察。它包括五個等級，開發的能力越強，開發組織的成熟度越高，等級也就越高。這五個等級的具體定義如下：

初級(Initial)：軟體發展過程中偶爾會出現混亂的現象，只有很少的工作過程是經過嚴格定義的，開發成功往往依靠的是某個人的智慧和努力。

可重複的(Repeatable)：建立了基本的專案管理過程。按部就班地設計功能、追蹤費用、根據專案進度表來進行開發。對於相似的專案，可以重覆使用以前已經開發成功的部分。

已定義的(Defined)：軟體發展的工程活動和管理活動都具備文件化、標準化的，它被集結成爲一個組織的標準的開發過程。所有專案的開發和維護都在這個標準基礎上進行制定。

被管理的(Managed)：對於軟體發展過程和產品品質的測試細節都有很好的歸納，產品和開發過程都可以定量地分解和控制。

最佳化的(Optimizing)：通過建立開發過程的定量反饋機制，不斷產生新的思想，採用新的技術來最佳化開發過程。

除了第一級，其他每一級都有幾個特別值得注意的關鍵過程。第二級的關鍵之處是建立基本的專案管理控制，他們是需求管理、軟體專案計劃、軟體專案的跟蹤和監督、軟體轉包管理、軟體品質保證和軟體組態管理。

第三級的關鍵是既關注專案問題，也關注組織問題，因爲組織建立起能促使高效率軟體工程制度化的基本架構，和跨專案的管理過程。它們包括組織過程關注程度、組織程序定義、培訓專案、集成化的軟體管理、軟體產品化機制、專案組的內部協調和對出現錯誤的複查。

第四級的關鍵是對軟體發展過程和軟體產品都有一個定量的理解。它強調的是定量的過程管理和軟體品質管理。

第五級則強調,不論組織還是專案必須追求持續的、可度量的過程改進。包括缺陷預防、技術更新管理和流程改造管理。

在這五個等級之下，有 18 個 KPA(Key Process Areas)，再來是 52 個 Goals，達成每個軟體成熟度時一定要做到這 52 個 Goals。而 KPA 只是提供達到 Goals 可能的做法。

3. Extreme Programming

XP 是由 Kent Beck 於 1996 年代初期提出。Kent Beck 是一個程式設計師，他以程式設計師的觀點看程式設計這項工作。長期以來，程式設計的工作受到開發期限（dead line）的壓迫，他們的產出往往無法達到應有的品質。XP 從人的本質考慮如何讓程式設計師，可以有尊嚴的工作，而同時可以產出高品質的軟體。XP 的精神可以說是實踐 Agile（中文譯為敏捷、輕量、靈活、輕快）軟體發展宣言的四項價值觀：『個人及相互交流勝過軟體工程和工具』、『可運行的軟體勝過完整廣博的文檔資料』、『與用戶合作勝過合約談判』、『回應變化勝過服從計畫』。

XP 提出四個核心價值觀:「溝通、回饋、簡化、勇氣」。從這四個核心價值觀，Kent Beck 為 XP 衍生出 5 項最優先的原理:

- **提供迅速的回饋**
  根據認知心理學的原理，得到回饋的時間和學習的速度有很大的關係，愈快得到回饋，學習的成果愈大。XP 將開發週期縮小，因此得到回饋的時間也縮短，若程式有問題的時間也可及早發現。
- **簡化**
  XP 主張，寫程式是用來滿足客戶的需求，不是用來讓程式設計師展示個人技術能力。因此，應使用能達成專案目標最簡單的技術來實作。XP 不主張為了程式未來的可延展性而使程式架構變複雜，因為程式需求必定不斷改變，若未來這項功能不再需要，則為了可延展性而做的努力也將白費。
- **漸進式改變**
  就像開車一樣，駕駛車輛時不可能不調整方向盤，加駛人必須隨時保持專注，一點一點改變方向盤的角度來配合實際路況。
- **擁抱改變**
  在軟體開發過程中，需求改變是一定會遇到的問題，通常也是程式設計師最煩惱的問題，XP 告訴我們必須預期改變必定會發生，透過開發週期變短，隨時和客戶保持接觸，一點一點調整方向，確保開發方向和客戶的需求同步。
- **做有品質的工作**
  XP 非常注重測試，甚至主張先寫測試程式碼，才開始寫主要程式碼。必須確保每一行程式碼都要被測試，唯有透過不斷嚴格的測試，才可以確保軟體的品質。

Kent Beck 並提出十二項務實的基本作法來落實 XP 的執行，他們是「規劃」、「小量發行」、「簡化設計」、「測試」、「持續性整合」、「程式碼重構」、「雙人程式設計」、「程式碼共享」、「每週 40 工時」、「客戶駐點」、「客戶與程式人員使用相同術語」及「遵行程式碼慣例」。

4. CMM/CMMI 與 XP 在小型專案的適用性

4.1 CMM 量身訂做(Tailoring and interpretation)的必要性

CMM/CMMI 制定模型時為求彈性，將整個規格以階層式來制定。如下圖。

Maturity levels	(5 levels)
→ **Key process areas**	(18 KPAs)
→ **Goals**	(52 goals)
→ *Key practices*	(316 key practices)
→ *Subpractices and examples*	(many)

最上層是 5 個軟體成熟度，接下來是 18 個 KPA(Key Process Areas)，再來是 52 個 Goals，達成每個軟體成熟度時一定要做到這 52 個 Goals。而 KPA 只是提供達到 Goals 可能的做法。一個 KPA 可能達到一個或多個 Goal。經常輔導業界做 CMM 的學者指出，他們依個別公司文化的不同，通常也會找到 KPA 之外的方式來達成 Goals[3]。在最下層(Key practices / Subpractices and examples)才是 CMM 規格書所提供的具體作法及範例，值得注意的是這些範例都是基於大型專案/團隊所提出。

基於 CMM 這種高度彈性的特色，在八種情況下，我們必需對 CMM 量身訂做(tailoring and interpretation)，以適合組織本身的運作方式[10]:

- 非常大型專案或組織。
- 虛擬專案或組織。
- 專案或組織分散各地時。
- 快速快發。
- 研發工作(R&D)。
- 維護工作。
- 軟體服務(Software Services)。
- 小型組織。

本文所關心的「小型專案/組織」及「實驗室單位」都屬於這種要量身訂做的類型。

4.2 CMM 與 Extreme Programming 如何相互配合

由於 CMM 只提供「要做什麼」(What to do)的原則性指導，並未嚴格規定我們是「如何做」(How to do)，所以在達成 CMM 目標時需要利用智慧(Intelligence)及常識(Common sense)來做判斷並因地制宜選擇最好的做法[5]。

根據研究指出，大部份的 XP 方法論所要達成的目標其實和 CMM/CMMI 在 Level 2、Level 3 及少數的 Level 4/ 5 所提出的是一致的[1]，如下圖:

Level 2 KPAs	Satis-faction	Level 3 KPAs	Satis-faction	High Maturity KPAs	Satis-faction
RM	√√	OPF	√	QPM	--
SPP	√√	OPD	√	SQM	--
SPTO	√√	TP	--		
SSM	--	ISM	--	DP	√
SQA	√	SPE	√√	TCM	--
SCM	√	IC	√√	PCM	--
		PR	√√		

√ partially addressed in XP
√√ largely addressed in XP (perhaps by inference) (in the appropriate environment)

換句話說，CMM/CMMI 告訴我們「什麼」才是正確的軟體開發，而 XP 則告訴我們要「如何」落實這些正確的軟體開發方式。由此可知，若我們能在小型團隊中實現 XP，等於部份達成了 CMM /CMMI Level2/3 的需求。這樣的成果相對於很多小型團隊原本毫無章法的開發流程來說，已經得到相當大的改善。

但即使如此，將 XP 落實並不容易。舉例來說，XP 的 Pair-Programming 是最引起爭議的一項規定。Pair-Programming 強調雙人共用一台電腦來寫程式，一個人開發時，另一個可以寫文件或構思，同時有問題時二人也可以互相幫助。在二人組中若有至少有一人是資深程式開發人員，這種方式是合理的，但若二人都是初學者，恐怕會有反效果。再者，程式開發人員即使是寫作文件，也都已經習慣用電腦來寫作，若強制二人共用一台電腦其實也是不合理的。

我們的建議是先行比較團隊本身文化及狀況，以漸近的方式來實施 XP 的部份機制。因為 XP 方法論曾引起很多爭論，強制將它所提出的方式套用到任意團隊中，不見得能達成最好的效果。

4.3 案例探討

2003 年寒假期間，政大資訊科學系的智慧型媒體實驗室(IM Lab)推行了一個實驗室網站的重構計畫，重構重點在於突顯 IM Lab 研究主題及研究成果以吸引更多優秀人才加入。

整個開發週期約為 10 天(約 2 週的工作天)，參加成員約 8 名，是一個典型小型團隊/小型專案的軟體開發。IM Lab 原有網站是以 J2EE 技術建構在 Linux 平台上，採用 Resin 做為應用程式伺服器。經過實驗室成員開會討論的結果，整個重構的重點如下:

- 針對使用者界面及網站 Layout 給使用者的觀感重新設計，能讓使用者將網站的 Layout 和 IM Lab 的特色聯想在一起。
- 加強網站上的文案寫作，詳細介紹實驗室的研究重點及成果。
- 實驗室成員個人簡單資料、研究專長及興趣應該可以在網站上供參觀者瀏覽，針對私密性資料，應該提供實驗室成員客製作(customize)的功能，能夠用自己的帳號登入、維護，並調整要公開的資訊。
- 應該加上留言版或討論區，以利參訪來賓發問、提出建議或成員討論、聯絡用途。
- 加強網站的安全性，防止入侵意外。

整個專案進行的特色歸納如下:

- **Pair-Programming:**

  上面列出的重點工作所需技術除了 J2EE 之外，還包含文案寫作、美工及 Linux 系統管理。有很多實驗室新進成員對於這些技術一點也不熟悉，他們希望能藉由參與這次專案來學習這些技術，因此我們採設 XP 所建議的 Pair-Programming 方式，將整個工作規畫為四個小組，一個小組由二人組成。這樣做的好處是初學者可以隨時請教熟悉者而未來人員流動時，在技術的交接上也比較不會出現問題。

- **隨時看到成果，快速得到回饋:**

  在二個星期的工作天中，開發人員除錯完畢之後，可以隨時將成果上傳到正式上線的網站上，每人至少每天要每天下午五點前要上傳一次，以確定今天有工作成果。五點之後 QA 組就可以開始使用網站並在針對網站給予建議。在短期內看到成果，可以隨時修正方向，而不致於在整個案子完成後，才發現做出來的不是我們要的。

- **平行開發，並想辦法証明自己的工作成果：**

  由於許多工作步驟有相依性，比方說要先做圖，文案才能放上去，程式才能套用在美工人員的圖上。在那麼短的開發週期中(2 星期工作天)，這樣做很浪費時間。因此我們強調每個人應想辦法能展示或証實自己的工作成果。如寫文案的若暫時還不能整合到畫面上，也要想辦法將工作成果以 pdf 或 html 等方式發佈給 QA 組。而開發元件的人，也要先寫單元測式碼來証實元件的可用性。

5. 未來研究方向

CMM/CMMI 提供一個良好軟體開發流程應該達成的目標，而 XP 則是針對小型開發團隊提出了實際的做法。但是執行面的具體做法，只有很少的文章討論。主要的原因是如果要針對這二種方法論提出有效具體做法的話，會使這種具體作法只適用於特定平台或技術。

目前業界的主要開發平台有 J2EE 及.NET 二種，在 J2EE 方面，已經有很好的具體作法被提出[6]，在實際輔助工具上則有專案建置工具 Ant，單元測試 JUnit 及 HttpUnit，壓力測試 JMeter。反之.NET 上並沒有相對應的具體作法或工具。

Microsoft 開發上提供了很優秀的快速開發工具(Visual Studio.NET)，在軟體開發的方法論上也提出過 MSF(Microsoft Solution Framework)，但 Visual Studio.NET 並未整合 MSF 的理念，且 MSF 也只提供原則性的指引，並未討論到具體作法，且 MSF 的規定很繁複，並不如 CMM/CMMI 般有彈性，並不合適小型團隊採用。

未來如果可以針對小型團隊及.NET 平台提出 XP(或 CMM)的具體可行方式，並提供建置、單元測試、壓力測試的相關工具配合，相信可以對採用.NET 解決方案的小型團隊帶來很大的幫助。

6. 結論

經由一系列的討論與分析，我們發現 CMM/CMMI 與 XP 這二個方法論並沒有很大的衝突。許多 XP 的核心觀念，正是落實 CMM Level 2/3 的具體做法。對於小型團隊而言，CMM/CMMI 告訴我們「什麼」才是正確的軟體開發方式，而 XP 則告訴我們要「如何」落實這些正確的軟體開發方式。

對小型及研發型團隊而言，通常對所謂的「規則」比較容易抗拒，團隊成員通常都認為自己在團隊中已扮演了稱職的角色，不需要什麼規定。CMM/CMMI 及 XP 協助我們在團隊紀律(discipline)及創意間取得平衡。由於開發周期(iteration)縮短，團隊成員可以馬上看出專案的成果，進而得到成就感，嚴重錯誤也可及早發現。在開發過程中所產生的各種完整的文件，也可以在開發成員流動時，不對專案造成過大傷害。小型團隊成員較少，在溝通及修改規則時比大型組織容易，因此我們認為 CMM/CMMI 若加上 XP 方法的協助，可以使小型團隊在改善軟體開發的過程中成效更高。

7. 參考文獻

[1] M. C. Paulk. “Extreme programming from a CMM perspective”. In XP Universe, Raleigh, NC, July 2001.

[2] Beck, K. Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading, MA, 1999.

[3] M.C. Paulk.” Using the Software CMM in Small Organizations”, Joint 1998 Proc. Pacific Northwest Software Quality Conf. and the Eighth Int’l. Conf. On Software Quality, pp. 350-361, October 1998.

[4] Beck, K. Embracing Change with Extreme Programming. IEEE Computer, 32, 10 (October 1999) 70-77.

[5] M.C. Paulk. “Using the Software CMM With Good Judgment”, ASQ Software Quality Professional, 1(3), pp. 19-29, June 1999.

[6] Richard Hightower, Nicholas Lesiecki. “Java Tools for Extreme Programming”, John Wiley & Sons,

December 15, 2001.

[7] C.Wrandle Barth. “When the CMM Shoe Doesn’t Fit:Tools for SPI on Numerous Small Projects”, Raytheon ITSS, Sep 2000.

[8] Mikael Lindvall and Ioana Rus, “Process Diversity in Software Development” ,IEEE Software, (July/August 2000) 14-18.

[9] A.Laryd and T.Orci. “Dynamic CMM for Small Organizations”, Proceedings of the First Argentine Symposium on Software Engineering (ASSE 2000), Sep 2000.

[10] Mark C.Paulk , “Effective CMM-Based Process Improvement”, Proceedings of the 6th International Conference on Software Quality,Ottawa,Canada,28-31 October 1996,pp.226-237.