軟體開發過程導入 DevOps 的組織架構與推動策略之研究
A Study of Organizational Structure Revolution and Strategies Promotion Implementing DevOps in the Process of Software Development

陳弘恩	吳欽誠	邱璟明
Hong-en Chen	Chin-chen Wu	Jing-ming Chiu
財團法人	財團法人	財團法人
資訊工業策進會	資訊工業策進會	資訊工業策進會
工程師	正工程師	組長
hogen@iii.org.tw	mason@iii.org.tw	jmchiu@iii.org.tw

摘　要

軟體工程包含軟體開發技術以及軟體專案管理，軟體開發技術包括軟體開發方法學、軟體工具和軟體工程環境。軟體專案管理包括軟體度量、項目估算、進度控制、人員組織、配置管理、項目計畫等(朱孝國，2006)，其主要希望透過流程制定、規範、文件範本將軟體開發過程中的相關資訊，包含專案財務、專案進度、人力資源進行系統性記錄，以期降低不可抗的未知風險。

組織在進行軟體工程開發規範導入時，非一味地將規範照本宣科地複製進組織內即為完成，而是需要從組織特性、人員素質、工作氛圍、開發專案特性進行評估並將欲導入的開發規範進行調整後，以小範圍試行後再逐一擴散，開發規範的關鍵成功謂之於「人」，避免透過高壓式導入造成人員反彈，以規範為本內化後應用於組織，才是成功唯一途徑。

本研究以 CMMI LEVEL 3 導入經驗為基礎，分析歸納軟體開發過程中所應遵循的「對的事」，以 DevOps 概念進行需求擷取、系統開發及程式撰寫、測試、部署期能以程式碼版本控管做為改善的開端，輔以自動化的程式建置、測試及部署加速開發流程中反覆執行的動作，減少人為介入的影響，降低人為偏誤所造成組織傷害及系統傷害。

關鍵詞：DevOps、組織變革、軟體工程、敏捷開發

Abstract

Software engineering includes technologies for software development and project management. The former includes methodology of software development, tools, and environment of software engineering, and measurement, development estimation, scheduling, human resource organizing, sub-projects are embraced by the later(Chu, 2006).

Software engineering are aiming to lower unexpected risks by information related to process and regulation forming, paper templates. In contrast to nerdily and reluctantly have an implementation in to organization, it must take its characteristics, educational level of staff, environment atmosphere, project profiles in to consideration and then conduct fine tunings to fit such unique organization from tiny spot which is taken effect to an extension of the whole one.

Figuring picture out for how to digesting regulations to be tactics and rebound must be avoided is key success factor.

In this research, we propose THE RIGHT THINGS and technical structure to carry out concept, DEVOPS. DevOps facilitate and speed up over all process in requirement acquiring, system development and programming, test and deployment. A beginning of DevOps, Source code version control, assisted with automatic program building, testing and deploying to reduce repeated actions is anticipated to cut off human-being biases to cause organizational and technical harms.

Keywords: DevOps, Organization Evolution, Software Engineering, Agile Development

前言

軟體工程包含軟體開發技術以及軟體專案管理，軟體開發技術包括軟體開發方法學、軟體工具和軟體工程環境。軟體專案管理包括軟體度量、項目估算、進度控制、人員組織、配置管理、項目計畫等(朱孝國，2006)，如軟體開發成熟度模型(Capability Maturity Model - Integrated, CMMI)。其主要希望透過流程制定、規範、文件範本將專案管理、需求擷取、軟體開發過程中的相關資訊，包含專案財務、專案進度、人力資源進行量化系統性記錄，以期降低不可抗的未知風險。

然而組織在進行軟體工程開發規範導入時，並非一味地將規範照本宣科地複製進組織內即完成，而是需要從組織特性、人員素質、工作氛圍、開發專案特性進行評估並將欲導入的開發規範進行調整後，以小範圍試行後再逐一擴散，開發規範的關鍵成功謂之於「人」，避免透過高壓式導入造成人員反彈，以規範為本，內化後應用於組織，才是成功唯一途徑。

系統頻繁的更迭中系統崩潰的風險也日益增加，再者系統開發及系統維運所處的環境差異性更是巨大，使得程式開發者以及系統維運者之間的溝通矛盾及鴻溝也會越來越深，造成開發者與維運者長久以來處於對立的景況。

2009 年 6 月 23 日，在加州 O'reilly Velocity 大會中，兩位 Flickr 工程師 John Allspaw 以及 Paul Hammond 發表了「10+ Deploys per Day：Dev and Ops Cooperation at Flickr」。在當時讓許多會議現場的開發工程師極為震驚，一天內超過 10 天的部署對於一般的開發、維運人員是無法想像的。John 和 Paul 認為重新塑模一套全新的軟體方法，整合開發團隊(Development)及維運(Operation)團隊，將彼此所擁有的資訊變的透明，將會是加速軟體部署和回復唯一方法(Development + Operation = DevOps)。

本研究將主要探討：

- 創新應用服務研究所在由 CMMI LEVEL3 轉移至 DevOps 時在軟體開發過程中建議進行的動作。
- 為實現 DevOps 持續整合及持續交付概念而構築的系統架構。

組織流程的變革

為彌補及改善開發者與維運者長久以來處於對立的景況，在需求擷取、系統開發及程式撰寫、測試、部署期能以程式碼版本控管做為改善的開端，輔以自動化的程式建置、測試及部署加速開發流程中反覆執行的動作，減少人為介入的影響，降低確認偏誤、認知偏誤所造成系統傷害。

本研究以創研所內部過去所導入 CMMI Level 3 之經驗為基礎，修正 CMMI 以往較為冗長、繁複、大量文件記錄等缺點，加速需求擷取到軟體開發及部署的循環，快速回應客戶需求，並以自動化的系統架構輔佐專案開發流程中人為介入的頻率，同時調整組織架構及重新定義規範，並修正「倡議」，讓維運人員能直接進入開發團隊，從專案開發初期即建立以維運觀點進行軟體開發。每一個組織都會因其組織文化、開發人員屬性及習慣而擁有不同的企業氛圍，因此在組織流程的調適上，雖然可藉由已導入 DevOps 開發文化的組織做為參考範例，但依然需要進行組織流程調整，其終使成為獨特的流程文化。

創研所的專案在開發屬性、服務屬性以及規模的離散程度偏異大，在參照 Postgresql, Openffice, Subversion 和 Spotify 四個 open soure project 在組織文化、需求確認及討論、功能(feature)開發做法後，針對創研所的開發文化、人員開發習慣、組織特性進行倡儀(suggestion)訂定並精鍊出表 1。專案經理或管理人員可依據專案規模，如雛形級專案、尚待進行服務概念驗證專案、產品級專案以及營運級專案四類級別進行判定進而選擇所需進行的倡議。

然而需強調的是在 DevOps 文化之中，並無「規範」而是採取信任原則，並不需要大幅調整已習慣的工作方式，DevOps 所強調的是軟體開發快速更迭的內化心態(mindset)，而不是照本宣科執行組織頒佈的規則。

表 1 依專案規模驗證其對應之軟體開發倡議

編號	DevOps的「對的事」	雛形級	服務驗證級	產品級	營運級
1	Define requirement		★	★	★
2	Requirement verification			★	★
3	Define functional spec		★	★	★
4	Function spec verification			★	★
5	Issue triage		★	★	★
6	Check out code	★	★	★	★
7	Write code	★	★	★	★
8	Peer review			★	★
9	Unit test			★	★
10	Submit	★	★	★	★
11	Build	★(手)	★(自)	★	★
12	Integrated Config Mgmt in test env		★	★	★
13	Integrated deployment in test env		★	★	★
14	QC validation		★(手)	★(自)	★
15	Release validation			★	★
16	Integrated Config Mgmt production env	★	★	★	★
17	Integrated deployment in Production env	★	★	★	★
18	Rolling back deployment				★
19	Application Monitoring			★	★
20	Automated Dashboards				★

開發流程的變革

DevOps 是一組過程、方法與系統的統稱，用於促進開發（應用程式/軟體工程）、技術運維和品質確認（quality assurance）的跨部門溝通、協同合作與整合，其強調軟體或系統開發人員與運營者之間的互動、協同作業及整合(Pant, 2009)。

DevOps 脫離傳統的軟體和系統發展方法，頻繁改變軟體功能，以迅速將改善方案投入生產。DevOps 與傳統軟體和系統開發方法不同，會經常變更軟體功能，以迅速地將改善的項目快速投入生產中。雖然在 IT 環境中頻繁地進行互動性變更，可能會在許多層面上危害服務和品質，但自動化和協調工具可降低，甚至可消除變更帶來的負面影響。

DevOps 流程能協助軟體或系統快速開發和

佈署，並透過自動化進行快速變更，持續改善品質、安全性和可靠程度，以此減緩所產生之動態環境帶來的影響。

有效的 DevOps 需要開發人員與操作者之間有相互合作的文化及心態。就 IT 環境而言，這包括運算、儲存和網路系統管理員及操作人員，需要加快籌備、設計、規劃、開發、執行、測試、提供與運作，才能從現有或新的基礎結構迅速建立並提供服務。在營運階段中，持續監視與改善都需要進行與上述相同的循環，因此「持續交付」這個詞彙就經常用來說明 DevOps 模式。

如圖 1 所言，經由開發人員在進行系統開發時所做的單元測試以及開發時所使用的環境紀錄進行建構管理(Configuration Management)為起始，從而著手測試環境及正式環境在系統運行時所需要的系統環境構築並以自動化觸發測試及部署腳本從而可以進行測試及部署。

讓軟體每一次的修改都是微幅的，每一次的錯誤都是清晰的，加上經由反覆不斷驗證的整合及交付，一致化開發人員及維運人員的正式環境，提升部署掌握度，降低軟體部署時未知的恐懼。

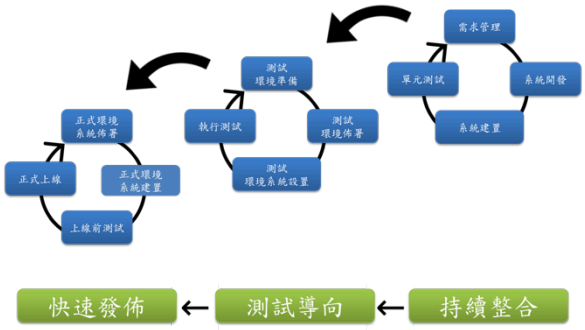

圖 1 應用程式持續交付流程

開源系統架構用以實現 DevOps 概念

本研究實做圖 2DevOps 系統架構。透過原始碼版本控管做為 DevOps 流程作動之始，與持續整合伺服器(Continuous Integration Server)進行連結(Hook)，當原始碼有變動時，立即作動測試腳本和建置腳本做為系統測試和建置流程依歸進行測試與建置，再行透過部署腳本進行系統及應用程式的部署，降低人為部署所造成的錯誤及延宕的時間，同時將建構管理(Configuration Management)一併納入原始碼版本控管的一部份。目的在於將部署時所需使用的基礎環境納入原始碼控管項目，讓開發者養成紀錄開發時的所在環境，讓維運人員了解應用程式及系統運行環境，達成開發環境、測試環境、部署環境皆是同一個環境，降低系統開發流程中因環境不一致所造成的測試、維運或部署的失敗，達到建立與維護工作產品之完整性

在系統及應用程式成功部署後，立即進行系統及應用程式日常運行紀錄收集，以視覺化方式呈現應用程式及系統的健康狀況，並且設定告警門檻值，以數據方式統計、分析及歸納系統運行瓶頸(bottleneck)，將後見之明(hind sight bias)、認知偏誤(confirmation bias)、結果偏誤(outcome bias)等人類情感所造成的決策偏誤以系統度量(metrics)降至最低。

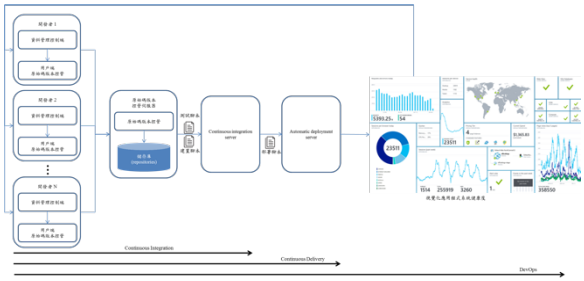

圖 2DevOps 系統架構

由於創研所專案數量、專案開發屬性及服務屬性、專案規模的離散程度偏異大，在使用 Jenkins 做為持續整合伺服器時，執行建置的次數頻繁，在建置(Build)後所產生的系統可執行程式(Artifact)的數量及檔案容量將會大量消耗整合伺服器的儲存空間。

因此本研究實作圖 3 可執行程式專用的儲存庫(Artifact Repository)系統，用以提供持續整合伺服器在完成系統建置之後做為 Artifact 存放空間。優點在於能把持續整合伺服器的計算資源與儲存資源分開，當在安排持續整合伺服器的資源配置時只需著重在其計算資源即可，而計算資料也是持續整合伺服器在執行建置時主要會使用到的資源；由於 Artifact Repository 系統具有高容量以及儲存資源容易擴充的特性因此，完成建置之後的 Artifactt 的儲存工作則由 Artifact Repository 系統來負責。

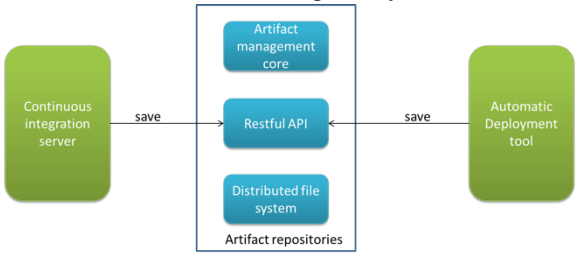

圖 3Artifact Repository 示意

在這個 Artifact Repository 系統共分為三層，分別是分散式檔案系統(Distributed File System)、核心程式(Artifact Management Core)與 Restful API 層，分別說明如下：

分散式檔案系統(Distributed File System)：此系統主責儲存大量 artifact，其本身所佔容量已非常巨大，因此組織內部建立一個具大容量且易於擴充容量的檔案系統，同時採用分散式檔案系統以符合系統需求。

核心程式(Artifact Management Core)：將實作管理 artifact 的功能。

API 層(Restful API)：提供與持續整合伺服器與自動化佈署工具之間的溝通方式。

Artifact Repository 系統是一個可獨立運作的系統，是 DevOps Pipeline 工具組中的一個重要工具。任何開發團隊可以將 Artifact Repository 系統

安裝於自己的開發環境中與持續整合伺服器及自動化佈工具整合，解決 Artifact 的存放問題。

結論

Devops 是加速軟體開發週期的流程變革，所談的是產品生命週期，此週期包含了需求、系統功能、專案期程安排、資源協調以及任何與專案相關的資訊、成員等皆涵蓋於此間(Walls, 2013)。

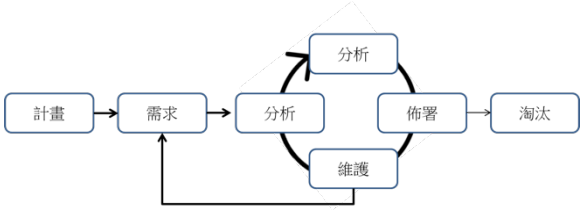

圖 4 資策會創研所軟體開發生命週期

透過軟體系統輔助，標準化軟體開發、佈署、維運，儘可能將圖 4 所示軟體開發過程中重複的行為自動化，軟體建置到上線營運速度同時增加服務維運方便性，縮短軟體開發生命週期所需要的時間，經由開發人員(Development)和維運人員(Operation)的密切合作，將軟體開發過程以及維運資訊透明化，以科學化數據輔佐決策，乃是 DevOps 的核心概念。

Gartner 於 2011 年所提出的 Hype Cycle for Cloud Computing 報告中提出 DevOps 是以敏捷(Agile)為基礎概念進而實現應用程式和系統的服務的遞送，並經由快速的反饋，修正服務的錯誤。

從技術面來看 DevOps，乃為從原始碼起以資訊系統進行納管，並透過建置伺服器(Build Server)，以測試腳本做測試，同時輔以部署腳本進行軟體部署服務，在這一連串的過程中，皆以自動化方式進行，減少人力介入，將認知偏差降低。

從組織面而言，規範是組織為簡化管理難度所訂定，在 DevOps 中，所有的流程皆以文化的方式內隱於專案成員心中，沒有繁雜的文件紀錄、過多的會議執行或是組織規範，所憑藉的是對於組織同仁的信任，並透過開發人員和維運人員的頻繁互動，瞭解雙方工作職掌，加速版本更迭，並以對方的權責觀點出發，將責任及風險共同分擔。

本研究以組織文化角度進行思考原點，提出 DevOps 導入所可能會遇到的文化衝擊以及軟體開發過程以不同專案規模所應該要進行的動作；同時實作自動化解決方案，大量縮短程式建置及部署時間，並透過數據度量降低人為偏誤所可能造成的系統傷害。

其它

- 本研究依經濟部補助財團法人資訊工業策進會「虛實整合智慧商務關鍵技術與平台研發計畫(2/4)」辦理
- 財團法人資訊工業策進會創新應用服務研究所(創研所)協助政府促成國內創新資訊應用的發展，透過核心技術的研究，設計與推動創新應用服務價值鏈，有效協助產業運用科技，創新應用模式以提升服務價值，使發展台灣成為全球應用服務創新櫥窗。

參考文獻

[1] Pant, Rajiv. “Organizing a Digital Technology Department of Medium Size in a Media Company, “ In http://www.rajiv.com/blog/2009/03/17/technology-department/. Last update 25th September 2010.

[2] Smith, David, *Hype Cycle for Cloud Computing*, Gartner Research, 27th July 2011.

[3] Walls, Mandi, *Building a DevOps Culture*, O’REILLY, 2013.

[4] 朱孝國，軟體工程簡介，取自：http://irw.ncut.edu.tw/peterju/se.html#cmmi，2006年07月14日