國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

開放原始碼軟體貢獻度分析

Accounting Contribution for Open Source Software Development

研 究 生: 張翊晉
指導教授: 黃世昆 博士

中華民國九十三年六月

開放原始碼軟體貢獻度分析

Accounting Contribution for Open Source Software Development

研 究 生: 張翊晉　　Student: Yi-Chin Chang
指導教授: 黃世昆 博士　　Advisor: Dr. Shih-Kun Huang

國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis
Submitted to
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
In Partial Fulfillment of the Requirements
For the Degree of
Master
In
Computer Science and Information Engineering

June 2004

Hsinchu Taiwan, Republic of China

中華民國九十三年六月

開放原始碼軟體貢獻度分析

研究生 ： 張翊晉　　指導教授:黃世昆 博士

國立交通大學資訊工程學系（研究所）碩士班

摘要

有鑒於目前開放原始碼軟體日漸普及，但仍欠缺完整的理論以說明開放程式碼與商業程式碼的品質差異。若以制度化的機制來檢驗開放程式碼專案，將有助於衡量開放原始碼專案的品質，亦可由各種不同相度去評斷一個專案的好壞。也希望藉此統計出專案發展者的貢獻度，以及開放程式碼專案之間互相利用的情形，以正比例回報所付出的貢獻度。我們將提出評估方法，以評斷專案的各項指標對於開放程式碼軟體持續度(sustainability)的影響，與貢獻度的分析,方便找出一般成功開放程式碼專案的成功因素。

目前對於開放原始碼軟體的發展現象，缺乏廣泛的理論據以解釋說明。我們就研究現況，解釋目前開放程式碼軟體的各種性質，提出修正且改進不合理的地方，包括取樣與分析的方法。

我們著重在驗證 SourceForge 上所選出的當月最佳專案，檢驗這些專案的發展過程，分析貢獻者(發展者/使用者)在程式碼(核心/使用者介面)上的貢獻度比例，並提出驗證假設”成立之初就有雛型的專案，其活躍值會高於沒有雛型的專案”。

1 簡介

1.1 研究動機

有鑒於目前開放原始碼軟體日漸普及，但仍欠缺完整理論說明開放程式碼與商業程式碼之間品質的差異。若以制度化的機制來檢驗開放程式碼專案，將有助於衡量開放原始碼專案的品質，亦可由各種不同相度去評斷一個專案的優缺點。也希望藉此統計專案發展者的貢獻度，以及開放程式碼專案之間互相引用的情形，以正比例回報所付出的貢獻度。我們將提出評估方法，以評斷專案的各項指標對於開放程式碼軟體持續度(sustainability)的影響，以及貢獻度的分析，以利找出開放程式碼專案的成功因素。

1.2 背景

”The Cathedral and the Bazaar”由 Eric Raymond 所提出，是開放程式碼研究中相當重要的理論，他提出兩種極端的現象。一種是像蓋教堂(Cathedral)的形式，由有經驗的工匠以集中的方式蓋好一座精緻的教堂，這種狀況下發展的軟體比較封閉；只靠一些人，集中在一起全力工作，工作過程中整個團隊的人只與團隊中的成員溝通討論，而不會採用團體之外的意見。另一種則是像市集的模式一般，每個人可以任意挑選自己想要的物品，每個攤販賣的物品都有不同的風格和價值，挑選東西之外，也可以自己發展有自己特色的商品，或者給予其他人一些建議以激盪出更棒的商品。簡單來說，一個是以集中管理來發展軟體，一個是以各取所需的方式發展軟體。自從 Eric Raymond 提出”The Cathedral and the Bazaar”的理論後，後續互有差異的理論接連發表，也使用開放程式碼的觀念日漸普及。

提到開放程式碼軟體，多數人都會聯想到Linux，以及以它爲基礎再發展的軟體。這是因爲Linux的出現之後，有人開始貢獻開放程式碼，並將程式碼置於

開放的網路空間上，其他人可以取用這些程式碼再做修改，影響開放程式碼社群的發展。OSDN(Open Source Developer Network)推出Sourceforge後，大家更能享受開放程式碼所帶來的便利，也可開發自己構想的開放程式碼軟體。SourceForge為一開放的網站，除了提供使用者下載專案的原始碼、安裝程式之外，也提供專案管理者以及發展者許多管理功能，例如Bug track、Patch等的forum以及CVS系統，讓發展者在開發軟體更加便利。我們的統計資料有極大部分都來取自這個網站。下圖Figure 1表示開放程式碼中專案開發分布的狀況。

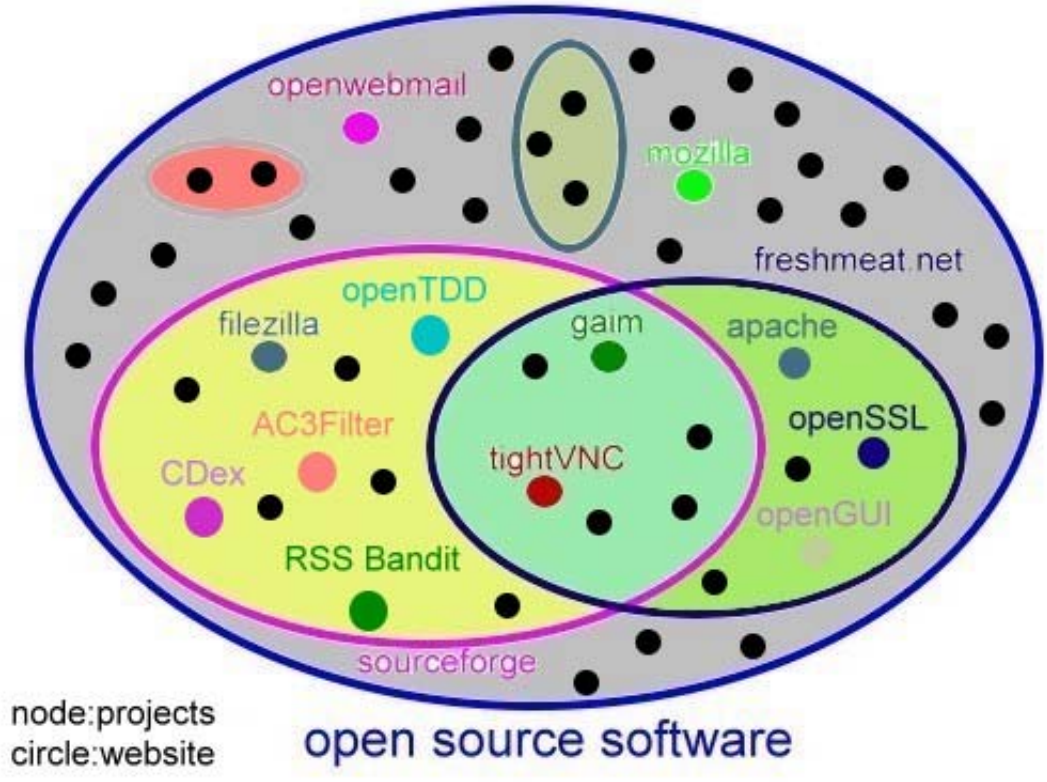


Figure1.Open Source Software　專案分布示意圖

在已發表文獻中[7,11,12,14,16]，對於開放程式碼與商業軟體品質優劣有甚多爭論，這裡將藉由參考論文提出的方法，加以融合改進分析。

1.3 研究目標

持續度分析(Sustainability)：開放程式碼社群中的成員流動頻繁，但不一定都會保持一定數量的人維持專案的成長與服務(support) 。在此我們提出持續度(Sustainability)的概念，持續度代表一個專案成長到一個階段後，期之後的表現均可維持在一定水準。最簡單的評比方式，是藉由比對當月最有潛力專案，其活躍值曲線的起伏，來檢視該專案是否有這樣的特性。

雛型(Prototype) ：若開放程式碼專案在註冊之初若有雛型模組(prototype)或有已一個開發團隊在運作，則可能加速這個專案的成長。亦即專案在 Sourceforge 註冊時就已有一個小的 prototype，藉此可以在短時間內就吸引夠多的使用者族群來使用，回報錯誤以及提供更多建議，幫助專案的成長，若符合這樣條件的專案，其活躍值可能會較其他一開始沒有雛型的專案略高。

生命力(vitality)以及歡迎程度(popularity) ：Sourceforge 中只用活躍度(activity)來代表了一個專案的活動力，但在 freshmeat.net 中則細分爲生命力(vitality)以及歡迎程度(popularity)。其中的生命力代表的是專案內部的活動情形，而歡迎程度近似於專案實際被外界使用的程度，這裡嘗試探討極端的例子，若單就 SF.net 提供的資訊來看，以目前有名 P2P 軟體的 Emule 爲例，就是屬於受歡迎(popular)但生命力(vitality)不高的例子。

核心與週邊關係(core-periphery)：我們嘗試探討開放程式碼發展過程，觀察核心與週邊之間的互動關係，例如核心設計者審核週邊使用者的臭重回報以及所進行的修正等，分析兩方的貢獻度，包括統計核心設計者專注於演算法的貢獻比例，與週邊使用者在於推廣專案的狀況。

2 相關工作

2.1 The Cathedral and Bazaar Theorem

這是由Eric Raymond所提出的理論，以教堂(Cathedral)象徵中央集權及責任劃分明確的商業軟體，以市集(Bazaar)象徵各取所需與責任劃分鬆散的開放程式碼軟體。其中提出的要點包括”Brooks' Law does not apply to Internet-based distributed development”、"Given enough eyeballs, all bugs are shallow"以及”Linux belongs to the Bazaar development model; The OSP model automatically yields the best results”等。這些論點發表後的相關討論相當豐富，持正反意見的論點都有。可將 Cathedral 以及 Bazaar 視爲一條線的兩端點，而不同軟體發展的模式則是坐落在這條線上的某點，因爲沒有軟體的發展模式是很極端地坐落在兩端點。開放程式碼的意思是”code is openly available”，但是另一方面的 closed source software 並不只侷限在商業軟體。而開放程式碼軟體也會有限制使用者與使用用途。可將”free”看成一種相度，而 OSS 是介於完全 free 到完全限制之間。

2.2 Sourceforge

SourceForge是已知最大的開放程式碼網站，至今已有86817個專案註冊以及910120個使用者註冊(截至2004/9/3)。豐富的專案數量，是研究開放程式碼的最大資料庫。首頁列出由內定公式計算所得的活躍度，對註冊的專案加以排序，以及下載數最高的前100個專案，每月還評選出當月指標專案。每一個專案都有不同性質的論壇(forums)，包括Bug、Patch、Feature Request以及Support Request等，以及CVS的archive以便於開發者發展軟體。統計資料上可列出最近七天、最近三十天或者以月爲單位的方式，列出該專案過去bug、patch、all tracker以及CVS comments次數的詳細資訊。

曾入選當月指標專案者，將是我們探討的重點。另外可藉由所列的活躍度排行，取得所有活躍度非零專案的資訊。

2.3 Lotka's Law and Power Law

Lotka'sLaw 是 1900 年初，Lotka 用以預測不同生產力的作者，數量的分布情形。由寫書的形式對應到開放程式碼中的生產力，來對開放程式碼中的作者，依照生產力區分。

($Ei=E_1 / i^n$)原本爲一量化生產力分布的數學式，也可預測每個作者未來的 authoring behavior。這裡不分析特定作者的生產力，而是針對"一群"作者。指數 n 代表不同 level 作者之間生產力的差異。若寫書模式的 n 與軟體寫程式的 n 相若，則可證明兩種 authoring pattern 是相似的。N 值越小，代表差異性越小。

Power Law用來描述一些領域的分部情形，也適用於開放程式碼的分布狀況。原本描述的領域包括英文字彙各個單字出現的頻率、地震強度出現的頻率、美國公司的規模大小、以及城市大小的統計等。主要的用途是轉換座標圖中的數據，採用對數座標之後，可得到近似線性的關係。

Figure 2,3表示取Log前與後的曲線。

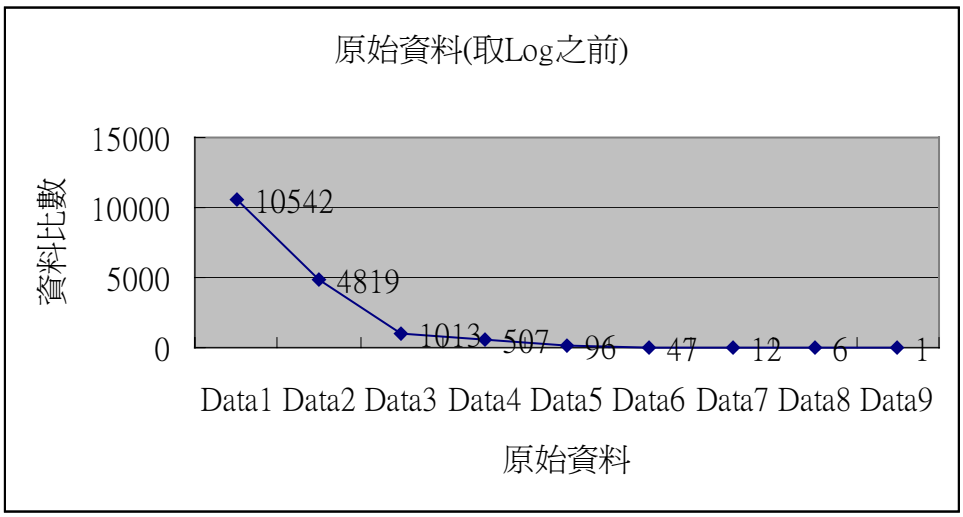


Figure 2.取log 之前的資料統計圖

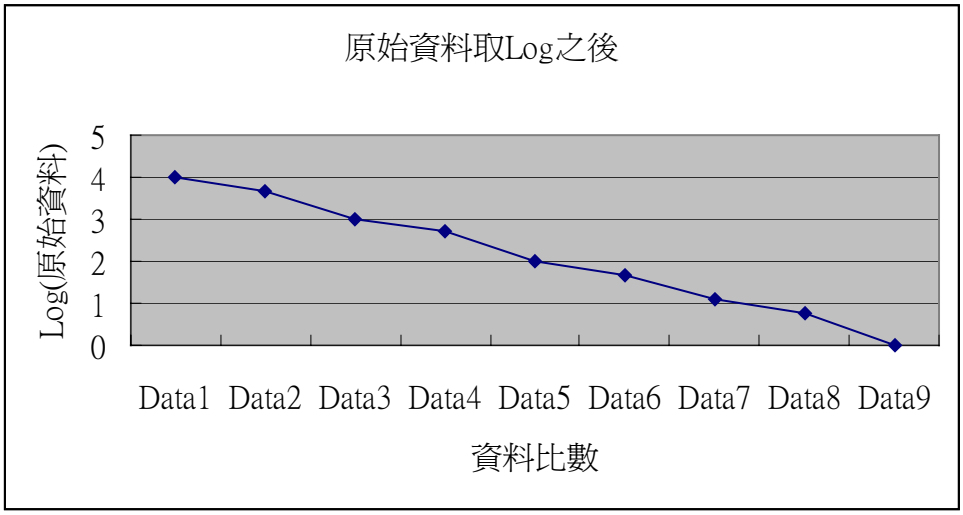


Figure 3.取 log 之前的資料統計圖

2.4 Brook's Law and Linus's Law

Brook's Law："Adding manpower to a late software project makes it later"，意指增加人力，卻無法使效率成正比增加。這個問題可以依 PCMM 在管理層面上做改進，以提升開發團隊整體生產力。

開放程式碼著名的理論"Linus's Law"—"give enough eyeballs, all bugs are shallow"，意即有足夠數量的使用者檢測軟體，就可以找所有的臭蟲。但由 Brook's Law 的角度來檢視，這樣卻是最浪費人力資源的方法，有許多人會花很多的時間去檢查一個臭蟲是否真的存在。

在 OSS 中，Linus's Law 必定是成立的，但 Brook's Law 是否會成立，尚需另外的討論。若只單就抓出 Bug 來看，多增加人力找出一個特定的臭蟲，不會拖累整體的生產力。但若牽扯到專案中的開發，又恰巧需要與其他人做一些溝通合作，Brook's Law 預測的情形就會發生，只要有兩個人以上做同樣的事，即滿足 Brook's Law。

另一方面，OSS 專案較商業軟體專案多的優勢，即擁有廣大的使用者(或者說 co-developer)，他們會在使用軟體的過程中發現臭蟲，使用過程中所花費的時間心力，是否算是" Adding manpower"也是可以深入討論的問題。

2.5 Cooking Pot Theorem

在OSS上的發展者，其貢獻的出發點不若在現實世界中，以可以獲取的回報多寡爲考量，而在可獲得的成就感這種非物質的回報。此理論的重點在使用者可到網路(視爲一個Cooking Pot)上，找出他需要的東西，不強迫使用者取得資源之後要有所付出。使用者使用後，視使用者自己的意志決定是否回饋，像回報臭蟲(bug report)以及對原始碼做一些修正等。簡言之就是各取所需，但並不強制要求要付出才可以獲得。在網路上面的付出，所得到的回報並不一定馬上顯現，可視爲一種長期的投資。或許現在做了一點貢獻，數年之後，回報可能是無可估計的。若只著眼在立即的回報，就是提出了一些解決問題的方法，而在此領域中得到他人信任，也就是所謂的名聲(reputation)。

2.6 PCMM(people capability maturity model)

PCMM 起源是爲了增加公司生存的機率，人員的流動率若過高，則公司無法有穩定的人力資源幫助公司成長，公司生存的機率與公司員工的流動率有絕對的關係，因此 PCMM 唯一套方法來減低人員流動率，並增加公司的整體競爭力。1998 年的一份報告指出，持續改善工作力(workforce)的公司，獲利會比沒有改善的公司高出 20%，也可以減少 7.05%的員工流動率。改善工作力的公司，其生存率與獲利都會有所提高。有很多方法可改善工作力，但仍有許多公司在施行上有困難或者進展緩慢，這是因爲還缺乏了委任管理的因素，而 PCMM 即是將改進工作力以及管理議題整合在一起的方法。

對於一般公司，有些東西是根深蒂固較難以改變的，像公司的文化等，要短時間內改變公司過去的習慣並不容易，於是 Humphrey 提出了漸進式的方法，逐步改善公司的文化以及行爲，並改變工作力常規(workforce practice)以增加公司的競爭力。

PCMM 是一套方法，以漸進方式加強管理，來改善發展團隊的內部管理。首先提升單一專案的管理能力，進而提升整體發展團隊的能力。其中提升專案管理

方面的準則，恰巧可以提供開放程式碼團隊依循的方向。PCMM 分五階段，從一開始沒有管理的第一層(initial level)開始，若可以成功地重複之前的範例(practice)來避免造成混亂的情形；便可進到第二階段的管理層(managed level)，此階段強調的是可以重複成功的範例，若每個單位(unit)做好管理的工作；即進入第三階段的整合層(defined level)，此階段中，會把較好的範例整合成一個程序(process)，若可以做到這點，組織即已建立起其專業的文化基礎；第四層是可預測層(predictable level)，能對程序所表現出來的績效(performance)量化，加以分析評估以對現有的範例做改進。應可重新產生(reproducible)，此階段可預測範例的績效；第五層是最佳化層(optimizing level)，強調有勇氣改變，改變管理(change management)變成一種不斷進行標準的程序，組織內部一直做改進，找出使組織獲利最多的程序以及減少常犯的錯誤來提升獲利。重點在強調管理階層一步一步找出影響專案的因素，希望將 KPAs 其中一些重要的 process 加以量化成幾個程度的指標，方便檢視專案的成熟度到了哪一個階段。

下圖 Figure5 代表 PCMM 在各層之間需要加強的能力

Figure5. PCMM 各個 level 演進的示意圖(from reference[2])

下表爲 PCMM 之 level 分類以及其中之 KPAs(Key Process Areas)

Table1.KPAs of PCMM

Level	KPAs
2-managed	staffing：分配工作、資源 compensation：給予名聲或實質的獲得 training and development：讓技能與工作可以相符 communication and coordination：爲未來的層級做準備
3-defined	competency analysis：確保成員能力仍有競爭力 workgroup development：增加團體的競爭力 e career development：有更好的職位讓人去奮鬥 competency-based practices：確保範例(practices)爲了增加競爭力
4-predictable	competency integration：把有競爭力的程序(process)做整合 mentoring：經驗傳承 competency based assets：competency process 相關的 asset，藉此增加競爭力 empowered workgroups：授權給 workgroup 對其負責的 business activity 負責 quantitative performance management：達到 measurable performance objects organization capability management：管理以及量化整個 workforce 的技能
5-optimizing	continuous workforce innovation：校準整體的績效 continuous capability improvement：提供基礎以增加競爭力 (在職進修等等) organization performance alignment：評估新的想法，找出最合適的實做

2.7 XP(Extreme Programming)

XP 是一開始不要規劃太多太深入，就目前的需要中，找出最急迫的一個，以最簡單的方法做。不同於一般軟體發展，需要分析很多東西，再完整地實做。這樣的方法適合用在變動性很大的開放程式碼世界，如果開發的時間太久，等到軟體推出的時候，通常產品已經跟潮流脫節了。

XP 的原理就是，簡單的設計(simple design)及簡單的基本測試(simple unit test)，讓重複(iteration)的週期變短，錯誤可以及早發現及修正，而非等產品出來之後，才發現產品規格已不符潮流了。

具體的活動(activity)方面，分成下列十二個：規劃遊戲(planning games)、小量發行(small release)、隱喻(metaphor)、簡單的設計(simple design)、測試(testing)、重組(refactoring)、雙人組程式設計(pair programming)、集體程式碼擁有權(collective ownership)、持續性的整合(continuous integration)、每週 40 小時工作時(40-hour week)、客戶全程駐點(on-site customer)、編寫程式碼協定(coding standards)。沒有硬性規定都要做到，只是做到越多越好。

XP 代表的另一個意義是工作都可以在數分鐘內完成，甚至一天之內可以做好幾個循環(iteration)。以下就是在數分鐘之內可以完成的工作。

XP 與一般軟體發展流程的不同在於，一般軟體發展模式是”do it right the first time”，而 XP 則是”do it right the last time”。XP 並不期望一開始就做到最好，而是隨著時間以及需求的變遷，慢慢調整接下來的步調並對未來的方向作細部修正。它強調的是將一開始詳盡規劃的時間省下來，多出來的時間用在後面的修訂上。一般認爲省下來的時間會比較多，因爲一開始的規劃並不一定與未來的趨勢契合，如果一開始就詳盡規劃，到後面要對程式碼進行大規模的修改反而更耗費資源。

2.8 XP 與 PCMM 之相關性

XP 的特色可助組織提升 PCMM 中的成熟度等級。XP 中的規劃遊戲(planning games)以及快速的改版(small release)可助組織縮短到達 PCMM 成熟度等級二(managed level)的時間。而到 PCMM 的 Level 3 之後，XP 較難提供有效幫助。因爲 PCMM 成熟度等級三強調整合以及管理，但管理的部分卻是 XP 最欠缺的，因此 XP 無法適用於即時系統(real-time system)以及虛擬團隊(virtual team)。

XP 可以針對每個專案不同的特性提供不同的常規(practices)，這也是他具有的彈性。XP 的溝通與簡單化也提供了 PCMM 所需的基礎。簡言之，XP 提供的常規，可讓使用者具備提升內部成熟度的基礎，但若想將組織的成熟度推高到下一等級的話，需用 PCMM 成熟度等級三之後的管理相關的程序(process)來幫助協助。

2.9 XP 與 OSS(open source software)之異同

相同之處是立即的回饋(rapid feedback)以及高頻率的釋出版本(frequent releases)。立即的回饋在 XP 方面是面對面溝通，找出最迫切的需求並完成，開放程式碼軟體是透過郵件名單(mailing list)作臭蟲報告(bug report)或是新功能要求(new feature request)。高頻率的釋出版本，XP 完成一兩個新的功能立即釋出，聽取顧客的評價及建議後，再做下一次的規劃並馬上實作，版本間的間隔很短。而開放程式碼軟體版本釋出時間間隔會比 XP 的循環(iteration)長，但仍比商業軟體的釋出時間間隔短。

XP 跟 OSS 不同的地方在於發展團隊的規模(team scope)以及溝通的方式(communication way)。團隊的規模上，XP 適用在小公司的規模，OSS 的規模常常是世界性的，沒有地理環境上的限制。溝通的方式，XP 強調的是立即回饋及面對面溝通，發展者跟顧客都可清楚表達自己意見，但 OSS 溝通的方式是利用 email 居多，且受到各地時區不同影響，效率上會比 XP 遜色許多。

簡而言之，在地理區域小的地方發展開放程式碼軟體，可利用 XP 的優點，來縮短釋出版本之間的間隔時間。下圖 Figure 6 表示了 Extreme Programming 與 Open Source 交集的部分以及不同的部分。

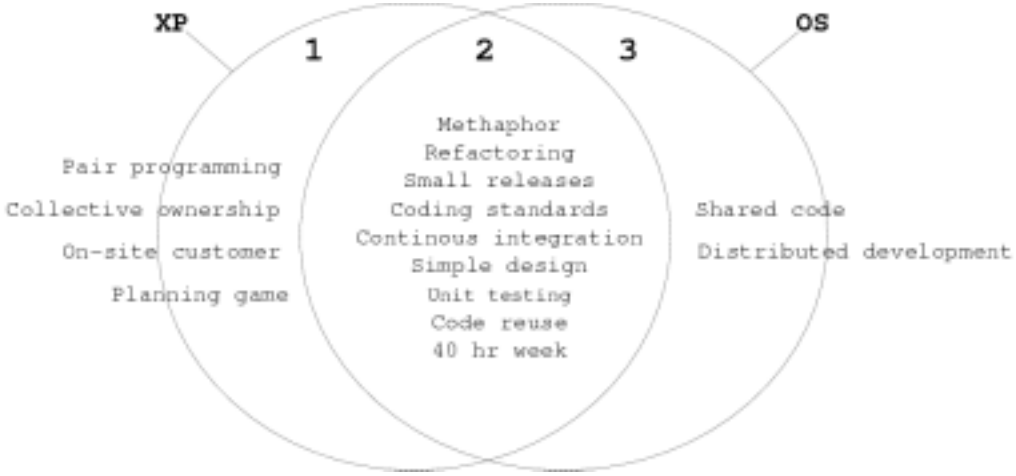


Figure 6 Extreme Programming 與 Open Source 的差異

2.10 Open Source Project and Open Source Community

一個發展良好的開放程式碼專案，通常會有一個組織完整的社群支持著。兩者的關係是相輔相成的，專案提供軟體給社群成員使用，而社群提供了一個環境供專案成員溝通交流。社群之中，最邊緣的使用者(passive user)，可以從單純的使用者，漸漸對專案的發展有興趣，開始做一些臭蟲報告以及臭蟲修正，漸漸晉升爲邊緣發展者(peripheral developer)。經過與核心發展者的溝通學習後，慢慢往社群的中心移動，甚至成爲整個社群的核心—專案領導者(project leader)。

一個人從邊緣到核心的角色進程如下：

passive user→reader→bug reporter or bug fixer→peripheral developer→ active developer→core member→project leader。一個人可以在社全中可以扮演的角色如下圖 Figure 7 所示。

Figure 7 開放程式碼社群中角色分布

社群是一個媒介平台，讓發展者在這平台上分享知識、經驗以及學習，而學習過程中所做的練習(practice)就會變成社群相關專案中的貢獻，因此社群成員是與社群相關專案的系統同步成長的。

影響學習的主因是社群的開放程度，越開放則成員獲取知識的管道越暢通，也有助於專案的運作。開放度的指標有兩個，一個是討論串的開放度，一個是原始碼的開放度。(如下表 Table2 from reference[9])

Table 2 開放度的指標

	Open release	**Open development**
Closed process	GNU; Jun	APACHE
Transparent process	Tcl/Tk	PostgreSQL
Open process		GIMP

Closed Process→討論串只有發展團隊內部(inner cycle)可以讀取並發表意見。Transparent Process→討論串發展團隊可以發表意見，但一般使用者(peripheral cycle)可讀取。Open Process→討論串所有人均可參與討論。Open Release→只有在版本釋出之後之後可以取的原始碼，即無法取得發展到一半的程式碼(interim developing version) 。Open development→隨時都可以取得原始碼(可由有無 CVS 判定)。

2.11 Open Source Software Developing Pattern

開放程式碼發展的模式分成了幾個種類：分別是探索導向(Exploration-Oriented)、實用導向(Utility-Oriented)以及服務導向(Service-Oriented)。

Exploration-Oriented模式正如Richard Stallman所說的"scientific knowledge to be shared among mankind"，較偏向research prototype，社群成員會互相分享所知道的知識，讓大家的創意可以互相激盪出更新的想法。由於最後釋出的版本只有一個，即使在社群中有許多功能被實做出來，若與釋出版本有所衝突就會捨棄；當然，也有可能會產生不同的分支(distribution)。有許多的功能會出現，但不是每個功能都會被收入釋出版本，在這樣的控制模式(control style)下，Exploration-Oriented是比較偏向Cathedral式的中央集權。

Utility-Oriented模式是偏向實用導向的，比較偏向Cooking Pot理論，想要什麼就到鍋子裡面拿的概念。使用者對取得的程式碼作修改，只是爲了滿足自己的需求，而沒考慮到與原始程式下一版本的相容性，所以這一模式的專案常常會有許多的分支(distribution)。此模式下，社群成員遇到問題時，會先上網去搜尋是否有現成解決方案，若存在則挑一個最好的來用，若沒有才自己嘗試寫出解答，當有許多解答存在時，最多人支持的解答會存活下來，這種物競天擇的現象稱爲比賽模式(tournament style)。

Service-Oriented模式以穩定爲優先考量，目的是讓使用者在使用時，不會時常遭遇到問題。這模式作修改會較保守，短時間內難有許多的改變。也因爲使用專案所研發的軟體的人數眾多，單一管理者很難決定要採用的patch或者new feature，取而代之的是以議會(council)的方式，讓大家來投票找出折衷的結果。

這三種模式的發展程度套用到上面Cathedral and Bazaar，總結的示意圖如下圖Figure 8：

Figure 8 不同發展模式的專案在開放度上的示意圖

以下的圖表 Figure 9 是取自 reference[10]。說明一個專案在不同時期，因應不同需求而以不同的發展模式進化。

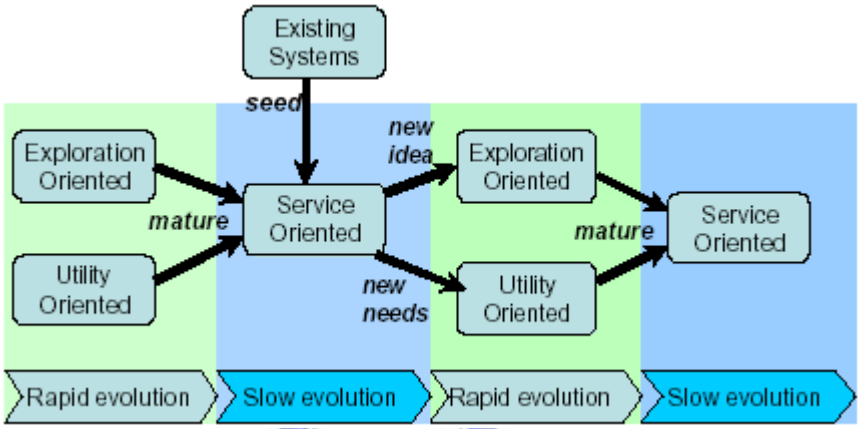


Figure 9 發展模式演進示意圖

下表 Table 3 是一些發展模式整理的表格(from reference[10])

Table 3 發展模式整理

Type	Objective	Control style	System evolution pattern	Community structure	Major problems	Examples
Exploration-Oriented	Sharing innovations and knowledge	Cathedral-like central control	Single branch Feedback from the community	Project Leader Many Readers	Subject to split	GNU systems Jun Perl
Utility-Oriented	Satisfying an individual need	Bazaar-like decentralized control	Multiple versions coexist Tournament style	Many Peripheral Developers Peer support to Passive Users	Difficult to choose the right program	Linux system excluding the kernel
Service-Oriented	Providing stable services	Council-like central control	Single branch Patches merged through control	Core Members instead of a Project Leader Many Passive Users that develop systems for end-users	Less innovation	PostgreSQL Apache

2.12 Motivation of developer and user to contribute

參與貢獻的人，都有其動機，而參與貢獻的動機不外乎是名聲與自身能力的提升。

其中名聲是 Reputation，不是實質上的物質利益，而是虛擬的聲望，貢獻者

可透過在社群裡面貢獻，漸漸提升自己的地位，往核心的部分接近。也可以透過貢獻，證明在某個特殊領域中有一定的能力，這對未來找工作會相當有幫助。

另一個原因是提升自己的能力，也就是透過 LPP(legitimate peripheral participation)，這個參與的過程在開放程式碼社群中發生。單純的使用者，可藉使用專案發展的軟體，透過網路與一些使用者交換心得，成爲進階使用者後，可針對專案發展的軟體提出建議。並取得部分程式碼做修正後再釋出，自己做修正，透過與其他發展者的溝通，對整個專案運作有更進深的了解，甚至加入開發團隊成爲發展者。成爲發展者之後，一開始由資深發展者帶領，逐步熟悉專案架構，以及撰寫程式碼需要依循的規則。慢慢地再往核心團隊接近，甚至成爲整個發展團隊的領導者。

上述的兩個原因，都不是著重在物質上的回報，而是滿足挑戰自我的渴望。

3 相關研究方法

3.1 Two Case Studies of Open Source Software Development: Apache and Mozilla

提出開放程式碼的多個特徵，包括工作不會委任給特定成員、沒有一定計畫以及缺乏傳統合作的機制等等。提出了兩大問題群組，第一組是把發展過程中的變數弄清楚，第二組是以使用者角度來看專案，著重在發展過程的結果。

先擷取 Apache 的資料之後，統計分析得到一些基本的假設，基本假設如下

- 由一個 10-15 個人組成的團隊來掌控整個發展過程，這個團隊創造了大於 80%的功能。
- 10-15 個發展者無法寫出超過 80%的程式碼，需要再將工作分細一點給子團體去開發。
- 在事件的發生頻率上，問題回報>缺陷修正>發展。
- 核心發展者可以創造新的功能，但是無法發現缺陷。
- 與商業軟體比缺陷密度，只有在商業軟體在釋出正式版本之前實作相比，Apache 的表現才會比較好。
- 發展者也是有經驗的使用者。
- 發展者對於使用者提出的問題總可以很快地回應。

等到也統計分析 Mozilla 的資料後，再驗證以上假設，只有前兩個假設需要作修正，得到下面的結論。

- 如果發展團隊，用非傳統合作方式進行工作，則團隊規模不會超過 10-15 個人。
- 如果專案夠大，則對於每個人責任劃分會更爲明確，要加入程式碼中的新程式碼需要被檢查審核。

用 perlscript 至 email list、CVS、BugDB 擷取 Apache 資料，用 perlscript 至

CVS 以及 BugDB 擷取 Mozilla 資料，利用 MR(modification request)取代 code added 較爲公平。將 Apache 以及 Mozilla 與商業軟體作比對。

優點是先分析 Apache 的數據，有假設後再對 Mozilla 做數據分析並驗證之前的假設。拿草稿給 core team 的成員看，公信力高。缺點是取樣的專案爲開放程式碼中成熟的專案，無法看出一般的開放程式碼專案與商業軟體比較的結果。無法看出 OSSF 中一般情形。

3.2 Characteristics of open source projects

藉由 freshmeat.net 上面的統計資料訂出公式計算各專案所得分數，另一方面也訂立指標的差距來分析。分析了各個領域中的專案數。定義出 stable 以及 transient developer，認爲影響一個專案的主要原因是 stable developer 的數量多寡。

將生命力(vitality)與普遍度(popularity)分開來計算，利用 freshmeat.net 上面的定義再加以改良。其中 popularity=((record hits + URL hits)*(subscriptions+1))^1/2，意義是將瀏覽數目與一些論壇(forums)討論的人氣，取其幾何平均數。而 vitality 的定義是(announcements * age) / (last_announcement)，其中的 announcements 與 releases 同義，其所代表的意義是接下來專案活動力的期望值。Announcements 就是全部 release 的次數，age 是專案註冊至今的時間，last_announcement 爲此專案上次發表 release 的間隔天數。

提出發展者團隊大小與專案大小無關的結論。將水平(horizontal)與垂直(vertical)的應用程式(application)分開來討論，水平的專案佔了 66%以上。水平的應用程式代表使用者會利用該應用程式來開發其他的軟體，像是系統開發軟體、軟體開發軟體以及資料庫開發軟體等等。而垂直的應用軟體則是像一般普通使用者使用的軟體，像是 filezilla 跟 gaim 這些軟體就隸屬於這部分。

優點是提出一些新的指標，像 app domain、doc level 以及 modularity level 對於生命力以及普遍度的影響。這是之前沒有提出過的想法。若將 developer 的 vitality 再做細分，不只看發展社群(developing community)的大小，有考慮到 Power

Law 造成的影響。缺點是取樣的數目不夠多(全部指取樣 406 的專案)，沒有一定的取樣原則。有些專案的間距過小，無法看出差異。未做到公平性與一般性。其中生命力的定義存在不小問題。

3.3 SourceForge default attributes

藉 SourceForge 上面的統計資料，經由公式計算，專案可得到活躍度的分數。優點是可以量化表示活躍度，讓使用者有一個基本的判斷依據。缺點是沒有將普遍度以及生命力分開來看，較不明確，無法得知哪個指標對哪一個影響較大，只能看出很粗淺的比較。

Sourceforge 是計算分數之後再列出百分比，但百分比無法看出專案之間的差異，只能當做篩選的標準，若可把計算所得到的絕對分數列出來，更能看出專案間的差異，或以第一名爲標準，其後的專案分數以百分比顯示約爲第一名的百分之幾(此爲 freshmeat.net 的方法)。

尤其活躍度降到一定程度時(像 90%以下)，專案的差異其實並不大，活躍度在 95%與 80%的專案差異程度，比活躍度在 80%與 50%的差異程度更大。這是因爲” Pareto distribution”造成的。下圖 Figure10 表示用百分比來表示活躍度造成的誤差。

Figure 10 活躍度百分比 V.S. 活躍度絕對值

3.4 The Perils and Pitfalls of mining SoureForge

針對如何分析 SourceForge，方法包含了三個階段：存取資料、分析、總結(spidering、parsing、summarizing)，並點出每階段需要注意的地方。Spidering(fetch data)的陷阱在於，要用 wait loop 防止被 ban IP，建議是將整個網頁都抓下再分

析。Parsing 的陷阱在於，不規則的行中斷(line break)，可藉由使換行符號變的可見來預防。以及如何找到正確的終止日期(close_date)方便計算貢獻度。Summarizing 的陷阱在於，對於 Nobody 這種匿名貢獻度如何做計算。它提供了四種看法，其中將每個 nobody 視爲單一個體較適合(這種做法跟把 nobody per item 或者 thread 算一個個體得到的數據差不多。)。另外有些欄位有不止一個值，也是處理的問題之一。

優點是可把臭蟲報告(bug report)中有實際貢獻(code written or regular expression)部分另外註記，可以找出真的有一些釋出程式碼(code submission)的臭蟲報告(bug report)。提出較有意義的觀點→如何將目前可以看到的一些指標(downloads 等等)的數目，反推回去得到他背後代表的意義。

缺點是沒有辦法看出所有專案的狀況，只能看出大略性的狀況。

3.5 On the Pareto distribution of Sourceforge projects

收集資料方法以及分析：

闡述 Open Source 分部的情形，與許多的領域相似，像是地震震度的統計、用字中英文字母的頻率、國民收入的分布等等，均是呈現特殊狀況與一般狀況的差異相當大的情形(”winner takes all”)。

評量標準以”下載”數爲基準。提出了兩個問題，其一爲資料的可靠程度，用人力去 check 其中五個專案，確定抓到的資料跟網站上的相同。其二爲是資料的可表現度(representative)，就是是否可代表整體的情形，他認爲 SF.net 的資料可說明 Open Source 的情形，即使有些最熱門的專案統計資料不在此(Apache、Mozilla 等)。

第一種統計方式是以整個 SF.net 的下載數目做分析，發現有個週期性，一個禮拜中的某幾天下載次數會最多，週末反而比平常日少，911 攻擊事件之後，有兩週的下載量跌到谷底算是特例。

第二種是以 30 天內，專案下載次數的分布。平均下來一個專案的下載數爲

70 次，但是分布很不平均(heavily skewed)。呈現了”winner takes all”的現象，如果將數據取 Log，會呈現很直的直線。一般而言，下載數少的專案，成長空間會下載數高的專案大。SF.net 下載數的成長有極大部分是由小專案的成長所貢獻。並有統計結果顯示，下載數少的專案易受到外力影響，比較不穩定。

優點是用實際的數據去顯示了一些大家都認爲很簡單不用證明的現象，但是至少比較有證據。提出的下載數有一個週期性是新的說法。缺點是只用單一相度來評斷有失主觀。不過可以大略看出普遍度。

3.6 The open source software development phenomenon：An analysis based on social network theory

分析發展者與專案之間的關係。認爲 OSS Community 有高度偏移(highly-skewed)的特性，若將兩個數值都取 log 後，可得到線性關係，稱爲”power-law”。基本假設爲 SF.net 的專案代表了所有的 OSS 的專案。

表現方式是將發展者視爲節點，參與同一個專案的成員會連成一個叢集(cluster)。分析的結果，最大叢集的人數爲 6862 個人，次大的只有 55 個人，只有一個人組成的叢集的總人數佔了所有註冊發展者的 25%。以電影工業來看，大約 90%的演員會被連結到一個極大的叢集，但開放程式碼社群還不到這樣的狀況，推測原因是因爲 SF.net 提供的開放程式碼社群發展得還不夠成熟所致。

14 個月的觀察中，最忙碌的發展者，所參與的專案由 14-27 個不等。只隸屬於一專案的發展者人數均維持在全體發展者的 80%左右。若以 top10 專案中的發展者來看，有 70%指屬於一個專案，但單一發展者，參與最多的專案數爲 12。

最後結果顯示，分析每個單一專案的人數統計(number of projects with N developers)，如果兩個相度均取 Log 的話，其分布可以找到一條極佳的直線。另外一個是以發展者參與的專案數來看(Number of developers on P projects)，也是取 log 的話，也可得到類似結果。如果以從集的角度來看，扣除掉最大的叢集(6862

developers)後的分布取 log，也可以得到一條直線。

優點是使用另一種相度(log-log)來檢視分布狀況，認爲 sourceforge 還發展得不夠久，沒有足夠時間讓發展者之間產生互動關係。用了兩種方法解釋(專案人數以及叢集人數)，會比較客觀。

4 研究方法

4.1 取樣

之前的研究中，最常遇到的挑戰是取樣的問題。這裡將取樣的問題分成兩個方面來討論，一種是隨機取樣，一種則是取出表現比較好的專案做統計。

如果要看一般性，也就是開放程式碼軟體中大略的狀況，必須採用大量的取樣，在這邊採用的方法，會強調一般性與分類。在一般性上，先假設不同的領域對專案品質的好壞影響較小，會用 random 選取的方式，以註冊專案在每個領域中的分類，依比例去各領域取相對數量的專案來計算，取得分數的平均，檢視持續度(sustainability)是否存在這些專案之中，統計出我們定義的”成功專案”佔了多少的比例，檢視專案的品質是否會受到領域不同的影響。

另外爲了取出品質較好的專案來與商業軟體作比對，取樣的條件參考” The Perils and Pitfalls of mining SoureForge”的做法，但是加以改良，選取”living”專案的條件降低一些。

原來的標準是超過 7 個發展者跟臭蟲報告(bug report)超過 140 個。超過 7 個發展者(實際上滿足此條件的專案非常少)，連最活躍的專案都有很多達到這個標準。bug report 要超過 140 個，這個部分受到專案年紀(age)的影響很多，年紀大的專案達到這個條件的機會越大。這邊想針對最近仍有活動的專案，所以訂最近一年內的一些指標來做篩選標準，另外加入下載數(downloads)的審核條件是因爲，有些專案下載數非常高，但是幾乎沒有臭蟲報告(bug report)，只單純利用 sourceforge 的下載功能而沒有利用到其他論壇(forum)的功能，但是因爲下載次數夠高，代表仍有進行更新的動作才有辦法吸引使用者前來下載。

改良之後選取的準則如下：

- Have release(必要條件)

- 3 個以上發展者(必要條件)
- 最近一年之內：平均每個月 bug 超過 3 個 or 下載數超過 100 個的專案

4.2 成功的定義(分爲 Vitality 以及 Popularity 探討)

Success 可以針對 activity 再細分成前面所提到的 vitality 以及 popularity 來做定義。這邊的公式是依據 freshmeat 中對於生命力(vitality)以及普遍度(popularity)的定義[10]加以改良。

原來的 $populairty = [(record\_hits + URL\_hits)(\text{subscriptions} + 1)]\text{^}(1/2)$ 這樣的問題不大，兼顧了兩個相度的測量，但是他忽略了代表普遍度最顯著的相度→下載次數，對某些人而言，甚至下載次數就可以代表一個專案的普遍度，sourceforge 也把下載的前 100 個專案另外列出，與活躍度最高的專案放在一起，可以看出下載次數的重要性。這邊提出的算法是 $[(record\_hits + URL\_hits)(subscriptions + 1)(downloads + 1)\text{^}(1/2)]\text{^}(1/3)$，就是將頁面瀏覽次數、論壇討論次數以及下載次數的平方根，再取幾何平均數。將下載次數先取開根號的原因是因爲下載次數多的專案與少的專案相差過大，而且下載次數遠多於頁面瀏覽次數以及論壇文章次數。

而 $vitality = \frac{release * age}{last\_release}$。以代號表示 R*A/L，存在著一個很大的問題。兩個專案 A、B，A 屬於較老的專案 Aa > Ba，在此狀況下，若 Ar=Br 且 Al=Bl，很明顯地，B 專案釋出頻率是高於 A 專案的，但是在此公式下，算出來的生命力(vitality)卻是 A>B。我提出的公式應改爲 $vitality = \frac{R}{A} * r$ (R/A)*r，過去 R/A 代表的數字是專案的釋出頻率(release frequency)，越高則代表此專案過去的生命力(vitality)越高，r 而代表的則是最近半年內該專案的釋出次數，可以看出近期之內專案的 vitality 狀況，兩者的綜合乘積，可以視爲該專案的生命力(vitality)指標。

將生命力(vitality)以及普遍度(popularity)分開可避免一個問題，那就是一個剛

興起的專案，在初期釋出版本的時候，可能會因爲與生命力(vitality)有關的指標，突然產生巨大的變動而使他生命力(vitality)值變高了，但是還不能此就說他夠受歡迎，因爲下載數量以及相關討論的數量還沒有真正浮現。將活躍度(activity)分爲生命力(vitality)以及普遍度(popularity)的好處在於，來獲取資訊的人可以依照其特殊的需求，來取得他想要的資訊。

所有專案以此公式計算出來的值，不管是生命力(vitality)或普遍度(popularity)，高於一個定值我就認爲他是屬於成功的專案。

4.3 雛型(prototype)的定義

過去的研究中，已知道年紀(age)與專案的活躍度相關性不大，這裡想進一步探討，如果專案一開始就有雛型(prototype)是否對於專案吸引外界使用者會有幫助？

關於雛型，一開始最簡單的想法：從專案成立的時間到版本釋出的時間間隔不到一個月，但是由 sourceforge 上面，無法直接取得相關資訊，不管是看所有下載的封包或新聞的資料庫(archive)，都會因之前舊檔案太多而清除掉一些古老的資訊，也有可能是 sourceforge 提供的空間有限。

第二種假設，利用目前專案中現有的發展者註冊時間與專案註冊時間比對，但是這個方法也行不通，因爲無法確定發展者加入該專案的時間是否就是發展者註冊的時間，加上專案以前的發展者可能已經不在發展者名單中，這方面的問題太多，也不予考慮。

最後採用的方法是利用 CVS log 的時間，拿來與專案註冊時間做比對。找出從專案註冊之後，在 2 個月內有超過 50 個 CVS comments 的專案，就是認定他是一開始就有雛型”prototype”的專案。

4.4 當月最佳專案的持續度(sustainability)

首先針對 sourceforge 上面最佳專案的部分進行是否可以有持續度(sustainability)的特性作比較，” sustainability”的定義是該專案當選過當月最佳專案之後每個月的活躍值，均維持在該專案當選最佳專案所獲的活躍值的一半以上。例如專案 A 在 2003 年 5 月得到的活躍值是 20 的話，如果要符合”sustainable”定義的話，從 2003 年 6 月開始到 2004 年 5 月這 12 個月份的活躍值都須超過 10 才符合。

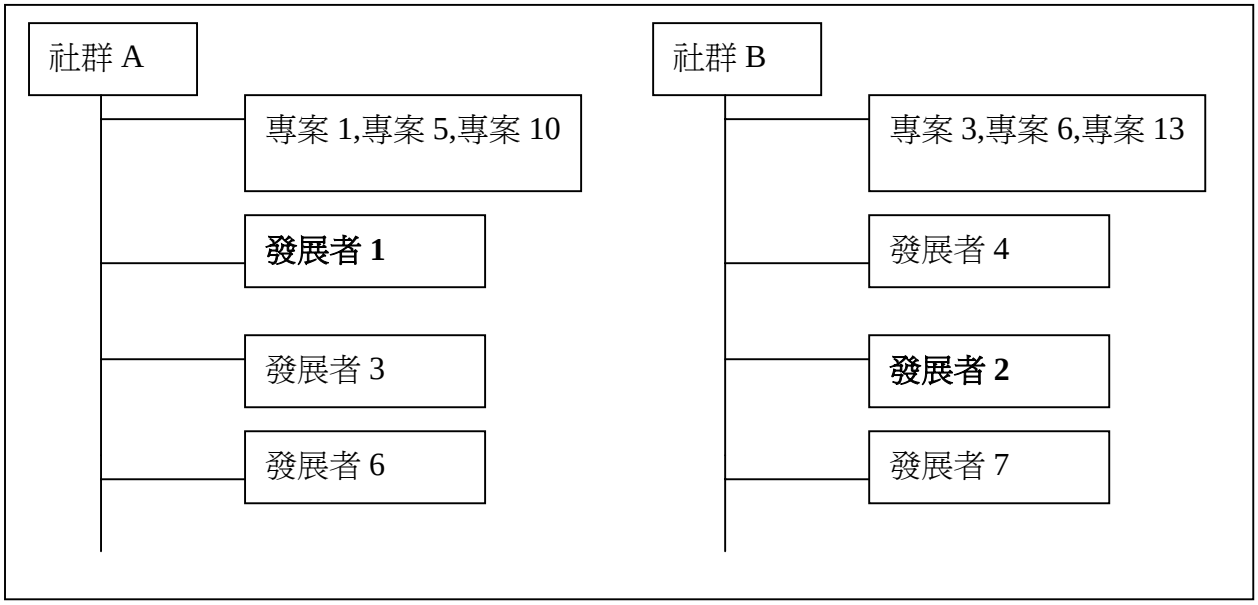
這邊活躍值是改良自 sourceforge 既定的公式，下表 Po 代表 forum posts，T 代表 task 數目，B 代表 Bug 數量，Pa 代表 patch 數目，Tr 代表 tracker item 數目，C 代表 cvs comments 數目，R 代表 release 數目，D 代表 downloads 數目，*activity_value=log (3 * Po) + log (4 * T) + log (3 * B) +log (10 * Pa) + log (5 * Tr) + log (C) + log (5 * R) + log (.3 *D)*。考慮到網頁上可以擷取的資料以及資料的重要程度，只取下列的指標計算 *activity_value= l log (4 * T) + log (3 * B) +log (10 * Pa) + log (5 * Tr) + log (C) + log (.3 *D)*。其中的 bug、support、patch 以及 all tasks 是以當月狀態設定成關閉的數量來計算，而 all tracker 多是屬於使用者討論部分，擷取當月新增的數量即可。

4.5 社群的大小

希望去驗證” The open source software development phenomenon：An analysis based on social network theory”這篇論文的說法是否正確，希望藉由再次統計發展者叢集(developer cluster)的大小來驗證。擷取的專案，希望是 activity 前 5000 名的專案，因爲 activity 太過後面的專案，有很多只是單一發展者註冊之後就沒有後續活動的。所以這邊只擷取前 5000 個專案並且統計在這 5000 個專案之中，發展者所構成的社群叢集大小(community cluster size)分布狀況大致上爲何。假設是

即使經過了兩年的發展(該論文發表時間是 2002 年)，仍不會如裡面所提及的，大部分的發展者會結合到一個最大的叢集之中。因爲電影工業與開放程式碼無法類比，電影工業的進入門檻要比註冊一個開放程式碼專案來的高。此外，電影工業之中的關係是永久的，只要曾經共事過，連結的關係就永遠存在，但是在開放程式碼中，發展者之間合作的關係是經常變動的。隨著時間的增加，電影工業會形成的叢集只會無限地增長下去，但開放程式碼社群形成的叢集卻有可能隨著某些成員的離去，而使叢集一分爲二。

採取的方法是，先由活躍度的排行上面取得活躍度非零的專案，藉由分析所有專案中發展者的帳號之後，如果發展者 A 同時有參與專案 B 與 C，則專案 B 與 C 的所有發展者則隸屬於同一個社群。



示意圖如下：

Figure 11 執行到一半程式所儲存的資料結構

現在讀取到第 N 個專案 X，發現此專案中同時有發展者 1 與 2，則將社群 B 合併到社群 A，原本的社群 B 則刪除。

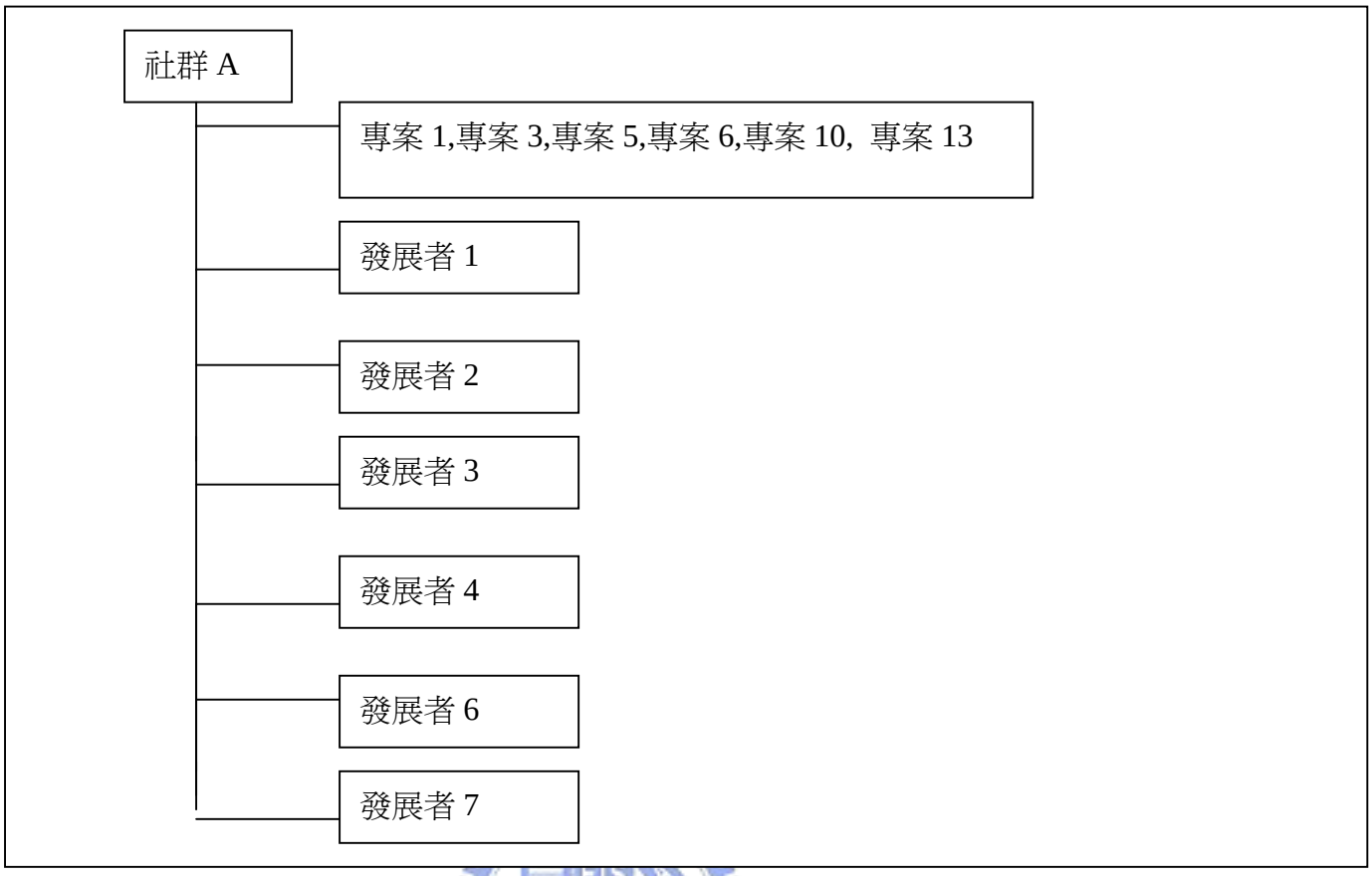


Figure12 讀取到第 N 個專案 X，將社群 A 與 B 合併

透過這樣的方法，我們可以自由地選擇活躍度在前面 1000 個或者 10000 個的專案，所形成的社群狀況，可以濾掉過多只有單一發展者的專案。

5 結果以及分析

5.1 當月最佳專案的持續度

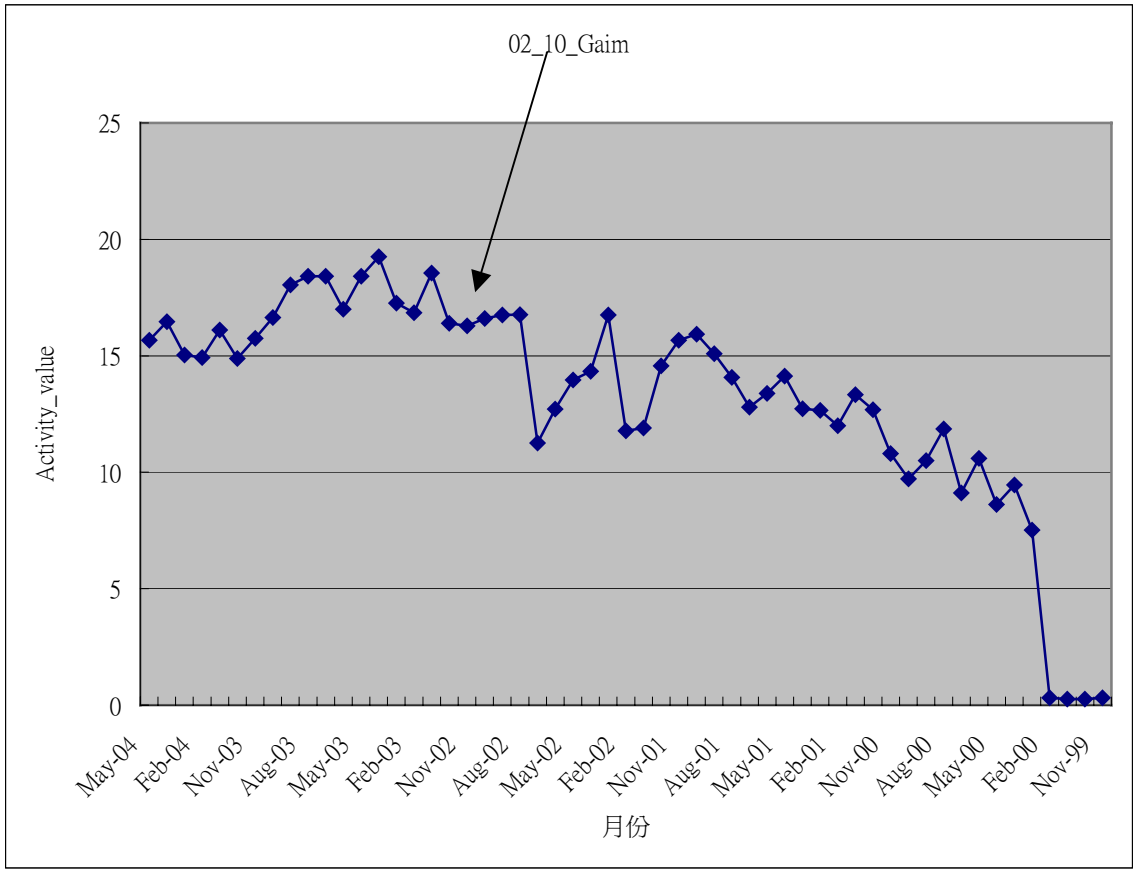


Figure 13 專案"Gaim"的活躍度曲線

上圖 Figure 13 是由研究方法中提出的公式，計算所得到的結果，02_10 代表的是 2002 年 10 月的最佳專案"Gaim"，X 軸爲時間，Y 軸爲當月該專案所得的活躍值(activity_value)，不同於 sourceforge 上面所提及的 rank 以百分比表示，以絕對方式呈現的活躍值較具意義。箭頭所指爲該專案當選當越最佳專案的月份，可以看出此專案的確具有持續度的特性，因爲當選該月的活躍值爲 16.6，之後每個月的活躍值都在 8.3 以上。

下列是曾當選過當月最佳專案的專案名稱：2002 年 10 月~12 月有 Gaim、

Fink、phpMyAdmin，2003 年 1 月~12 月有 SquirrelMail、Crystal Space、TUTOS、JBoss、POPFile、MegaMek、Tiki、BoaConstructor、TightVNC、gallery、filezilla、PhpGedView，2004 年 1 月~4 月有 phpBB、compiere、Mailman、BAFlag。

下表 Table 4 列出專案當選之後，每個月的活躍值，橫的 2002/10 代表 2002 年 10 月的專案，直的一欄代表的是該專案逐月的活躍值。當選當月最佳專案的活躍值以粗體表示，而之後若有低於 1/2 的情形則以粗斜體表示(例如 2003 年 2 月最佳專案 Crystal Space 在 2004 年 3 月的活躍值為 4.6，低於當選最佳專案時活躍值 11.1 的一半)。

Table 4 SourceForge 最佳專案活躍值統計

	2002/10	2002/11	2002/12	2003/01	2003/02	2003/03	2003/04	2003/05
2004/05	15.6	8.8	12.2	9.8	7.1	6.1	9.6	7.3
2004/04	16.4	10.0	11.4	9.3	6.9	8.2	10.7	8.2
2004/03	15.03	9.2	13.0	8.9	*4.6*	8.8	11.7	10.8
2004/02	14.9	8.2	12.4	9.0	*4.1*	7.9	10.4	9.2
2004/01	16.1	8.5	11.1	8.9	*5.0*	8.0	11.6	8.8
2003/12	14.8	11.5	13.7	9.3	6.3	7.8	10.4	9.05
2003/11	15.7	9.1	11.6	9.2	*4.1*	7.5	10.6	10.0
2003/10	16.69	10.8	13.5	8.8	6.8	6.9	11.7	11.61
2003/09	18.0	11.6	13.2	10.8	10.2	9.3	13.5	10.7
2003/08	18.4	11.9	15.2	10.7	11.0	10.5	12.5	12.2
2003/07	18.4	11.0	16.4	14.3	10.2	11.1	14.6	12.2
2003/06	16.9	13.6	16.4	12.7	11.5	9.9	14.3	13.0
2003/05	18.4	12.6	16.3	12.4	12.4	10.8	16.8	**12.7**
2003/04	19.2	13.8	16.1	13.6	9.6	11.4	**16.2**	
2003/03	17.2	13.9	16.6	12.1	7.2	**10.9**		

2003/02	16.8	14.4	17.2	12.1	11.1			
2003/01	18.5	13.8	14.6	15.8				
2002/12	16.3	13.7	14.6					
2002/11	16.2	13.7						
2002/10	16.6							

	2003/06	2003/07	2003/08	2003/09	2003/10	2003/11	2003/12	2004/01
2004/05	11.0	10.5	*4.4*	8.5	12.1	11.2	8.5	6.9
2004/04	9.6	10.4	6.1	8.5	12.7	9.3	9.6	6.9
2004/03	9.6	9.5	*4.2*	12.2	12.5	9.7	9.6	7.2
2004/02	9.1	12.6	5.4	9.8	10.3	10.3	9.2	6.9
2004/01	11.3	12.1	5.1	11.0	10.4	11.2	10.8	6.9
2003/12	9.0	10.7	7.3	7.7	10.9	9.0	10.8	
2003/11	9.2	11.3	7.0	6.6	11.0	9.9		
2003/10	10.3	12.3	7.2	11.2	12.5			
2003/09	10.9	13.5	7.1	9.1				
2003/08	11.3	16.1	9.2					
2003/07	12.9	16.1						
2003/06	12.9							

	2004/02	2004/03	2004/04
2004/05	13.2	6.9	9.2
2004/04	13.7	7.0	8.8
2004/03	12.4	6.7	
2004/02	12.3		

綜合以上結果，19 個專案中只有兩個專案分別在六個月份沒有達到持續度的定義，但再仔細分析，2003/02 的最佳專案 Crystal_Space 從當選到沒有符合定義的時間間隔有 8 個月，而 2003/08 的最佳專案 BoaConstructor 從當選到沒有符合定義的時間間隔則有 6 個月，也可以算是有符合持續度的定義，因爲我們可以預期的是，所有專案不是永遠都會進行活動，只能預期他活躍的程度，在一定時間內到達一定的標準以上。

另外附上活躍度在 90%左右的兩個專案活躍度的分數曲線。其中的 90%是指在最後一週的活躍度，不是代表當月的活躍值，不過可約略看出一些狀況。

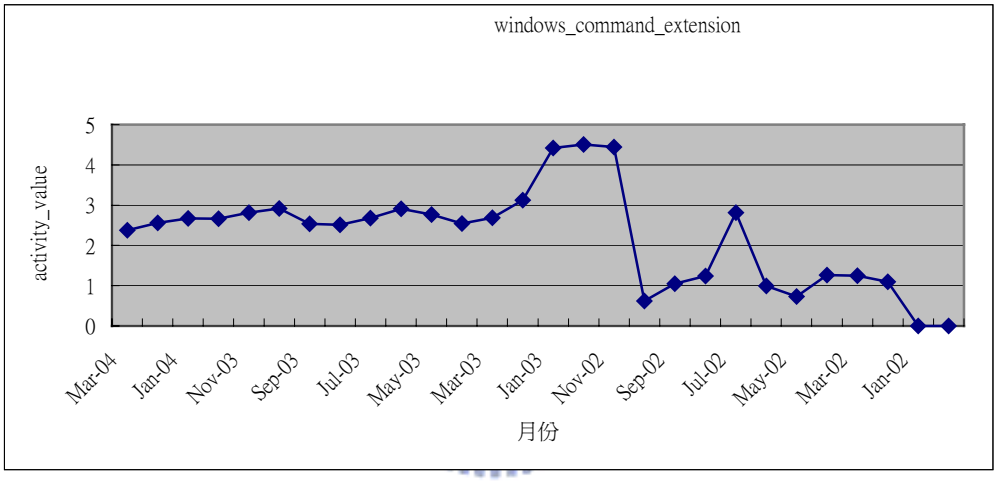


Figure 14 活躍度 90%專案”windows command extension”

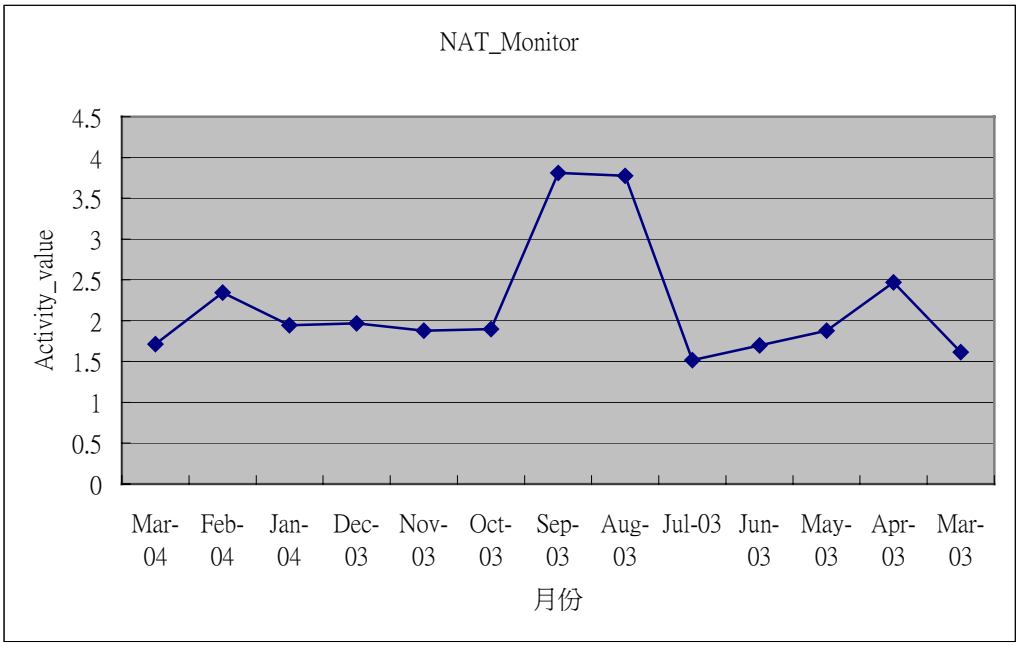


Figure 15 活躍度 90%專案”Nat_Monitor”

活躍度 50%左右取樣專案的活躍值曲線如下。

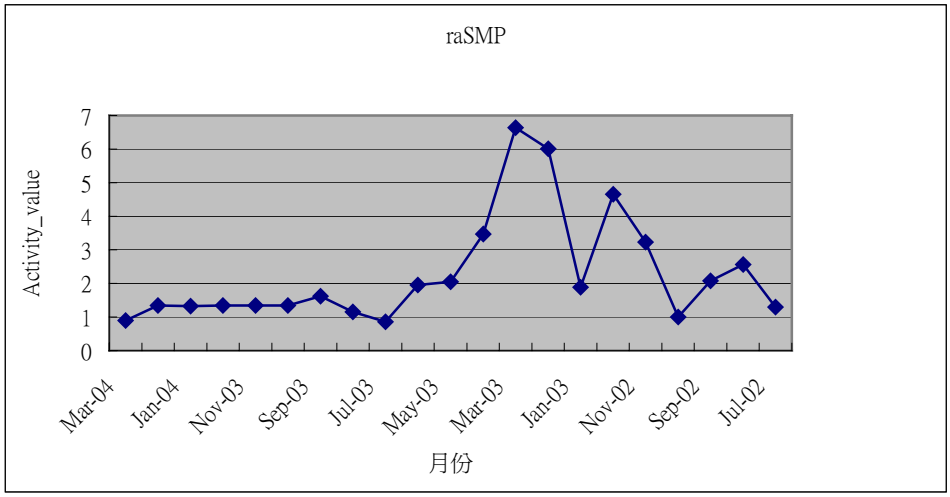


Figure 16 活躍度 50%專案”raSMP”

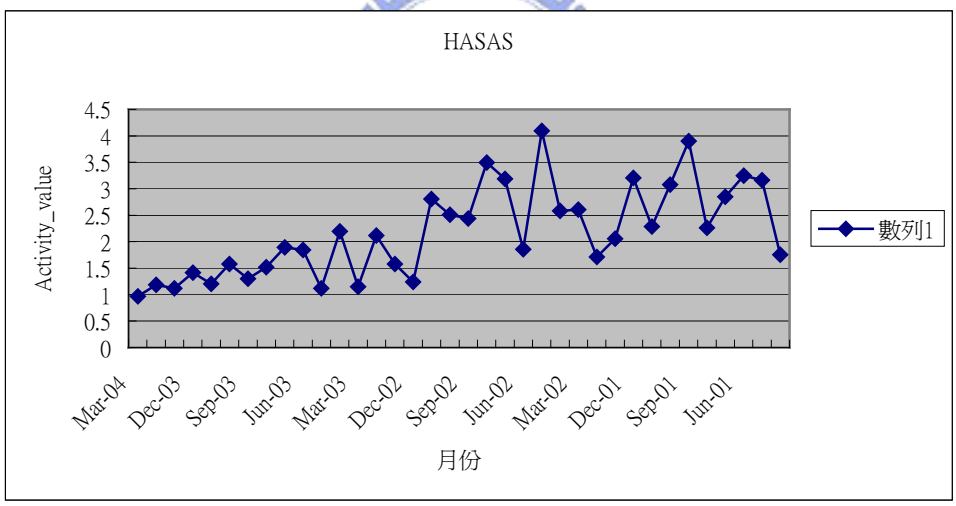


Figure 17 活躍度 50%專案”HASAS”

活躍度 30%左右取樣專案的活躍值曲線如下。

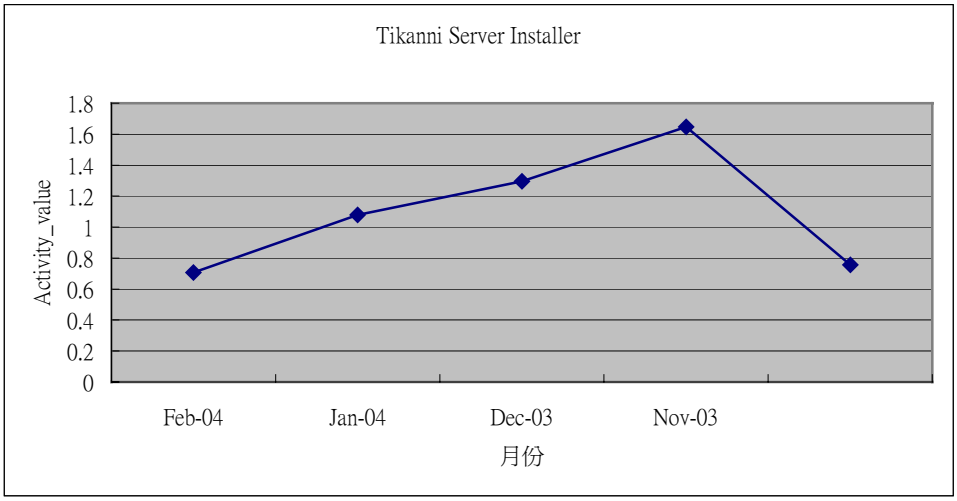


Figure 18 活躍度 30%專案”Tikanni Server INstaller”

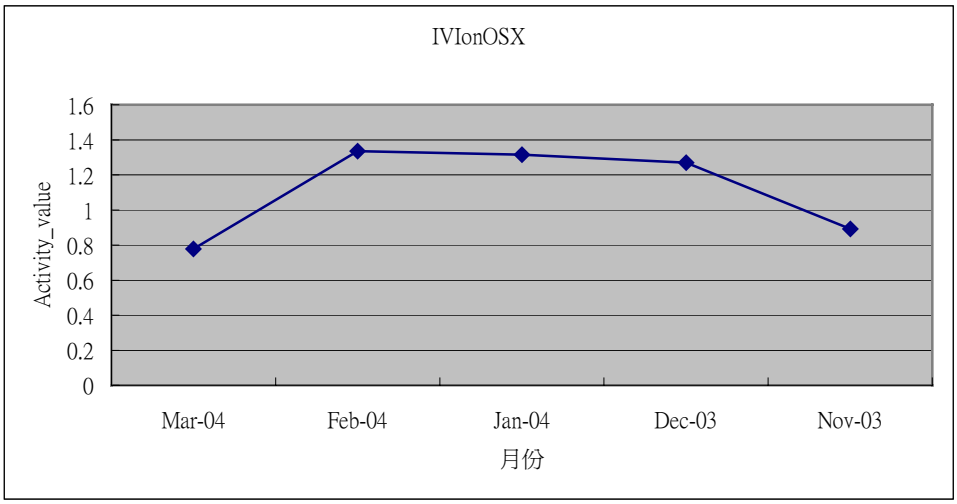


Figure 19 活躍度 30%專案”IVIonOSX”

綜合以上，如果最後一週的活躍度可以約略代表最後一個月的活躍度，我們可以看出 90%活躍度的專案活躍值大約在 2~3 之間，而 50%活躍度的專案活躍值大約在 1~2 之間，而 30%活躍度的專案活躍值大約在 0.5~1.5 之間。

Table 5 activity rank 與 value 對應表

Activity Rank	>99%	90%	50%	30%
Activity Value	8~18	2~3	1~2	0.5~1.5

以上六個專案與得到最佳專案來相比，可以知道以活躍值的多寡來判斷一個專案的活躍程度才是有意義的，只單純由百分比來判斷會產生極大的落差。也可觀察到開放程式碼中的”Pareto distribution"—活躍度在 95%以上的的專案跟 80%

以下的專案差距會很大。

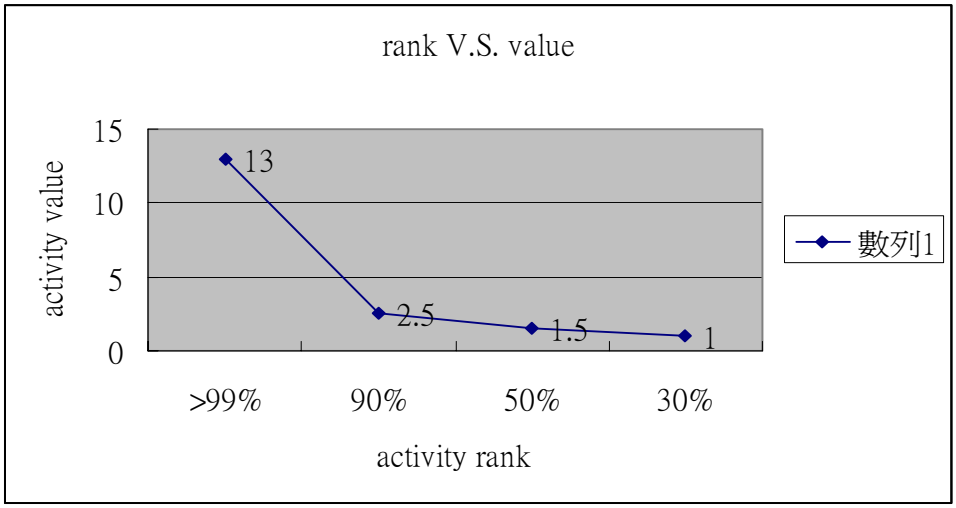


Figure 19 activity rank V.S. activity values

5.2 貢獻度的分佈比例

活躍度太低的專案，可觀察到的資料量過少，這邊選取的樣本是 FileZilla、phpMyAdmin(前兩個都當選過當月最佳專案)以及下載數相當多的專案"FCKeditor"。希望透過分析這些專案，可以幫助了解目前 core team/user 在 UI/kernel 的貢獻度比例。

每個專案設置的論壇不同，但基本的像是 Bug、Support Request、Feature Request、Patch 都會有，這三個專案也是有這些基本的論壇裡面去做統計。這邊採取的方法是以人工的方式下去看，會較爲精準。擷取資料的間隔設定爲以上三個專案最近的兩次釋出本板之間，論壇上的討論主題增加的數像，並確定這些討論中的貢獻已經實際加入到專案之中。有些專案裡面會有些特別的狀況，像一般觀念中，已經解決的問題會把臭蟲的狀態設定爲"Closed"，而一些比較屬於使用者個案的問題或者又再度重複的問題會設定爲"Deleted"或者"Duplicated"，但在 phpAdmin 裡面，有許多臭蟲報告(bug report)狀態被設定爲"Closed"，可是他的 resolution_id 卻是"invalid"、"duplicated"或者"wont fix"。像這樣的討論都不會列入實際貢獻度的考量。

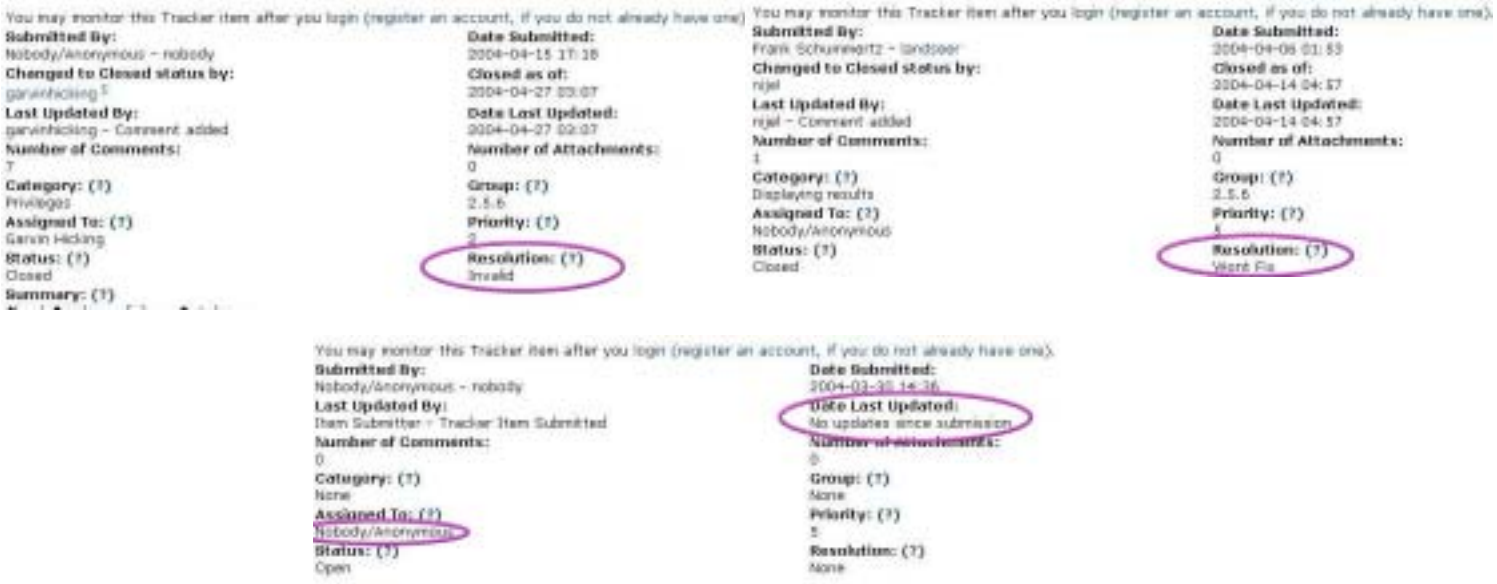


Figure21 不列入考量的報告範例

以下的貢獻度分成兩種，一種是實際是在專案中已經加入程式碼更新，一種則是有效的討論，啓發發展者作接下來的貢獻。

FileZilla

兩次 release 間隔時間在 2004/3/21—2004/4/03 之間。Bug 部分總共有 13 個，其中 open 有 11 個，closed 有 2 個。有 6 個 support request，狀況全部都是 open。這段時間內沒有 patch。全部 12 個 feature request，狀況都是”open”。

Table 6 FileZilla Contribution Proportion

實際貢獻度比例

	Core team	User
Kernel	1	1
UI	0	0

提出有意義的 bug、support request 以及 feature request，UI 與 kernel 的比例

	Kernel	UI
Bug	2	0
Support	2	3
New feature	1	9
Total	5(41.67%)	12(58.33%)

FCKeditor

兩次 release 時間爲 2004/1/29~~2004/4/4，擷取資料時間設定爲 1/30-4/3。Bug 全部 15 個，Open 的有 10 個，Closed 有 3 個，Deleted 有 2 個。Support Request 總共 25 個，其中 open 的有 12 個，closed 的有 12 個，deleted 的 1 個。有 4 個 patch，全部爲 closed。Feature Request 全部有 7 個均爲 Open。

Table 7 FCKeditor Contribution Proportion

實際貢獻度比例

	Core team	User
Kernel	11(55%)	1(5%)
UI	8(40%)	0(0%)

提出有意義的 bug、support request 以及 feature request，UI 與 kernel 的比例

	Kernel	UI
Bug	2	2
Support	5	3
Feature	2	2
Total	9(56.25%)	7(42.75%)

phpMyAdmin

Release 的間隔爲 3/01(2.5.6)-->4/21(2.6.0)，取樣的區段爲 3/2→4/20，值得注意的是他有釋出 2.6.0 的版本開放試用，並加以修正 bug。這段時間內總共有 52 個 bug，open 有 32，closed 有 14，deleted 有 6。Patch 只有一個爲 Closed。Feature Request 全部 25 個來看，可以列入貢獻評比的只有 3 個，有效的討論有 4 個，可能是因爲此專案的 popularity 較一般專案高，使用者素質相差過大，所以會有一些比較離譜的需求。

Table 8 phpMyadmin Contribution Proportion

實際貢獻度比例

	Core team	User
Kernel	12.5(43.1%)	5.5(19.0%)
UI	7(24.1%)	4(13.8%)
Total	67.2%	32.2%

有效的討論

Kernel	UI
5	4

綜合以上三個專案的結果，我們可以得到下表的結果

Table 9 Contribution Proportion of 3 projects

實際貢獻度比例

	Core team	User
Kernel	24.5(48.04%)	7.5(14.71%)
UI	15(29.41%)	4(7.84%)
Total	39.5(77.45%)	11.5(22.55%)

有效的討論

Kernel	UI
19(45.24%)	23(54.76%)

由於以上的 core team 只侷限於專案首頁所列的發展者，無法判別在討論中被列爲使用者的，是否是屬於與專案發展者屬於同一個社群之中，如果有這樣的情形，core team 所佔的貢獻度比例會再提高。

5.3 雛型影響活躍度的程度

取樣的專案由活躍度排行榜上擷取，取樣的範圍在 99%、96%、93%、90%、85%、80%、75%、70%、65%、60%、55%、50%附近的 10 個隨機取樣專案。找出每個範圍中，滿足 prototype 定義的有幾個。結果如下：

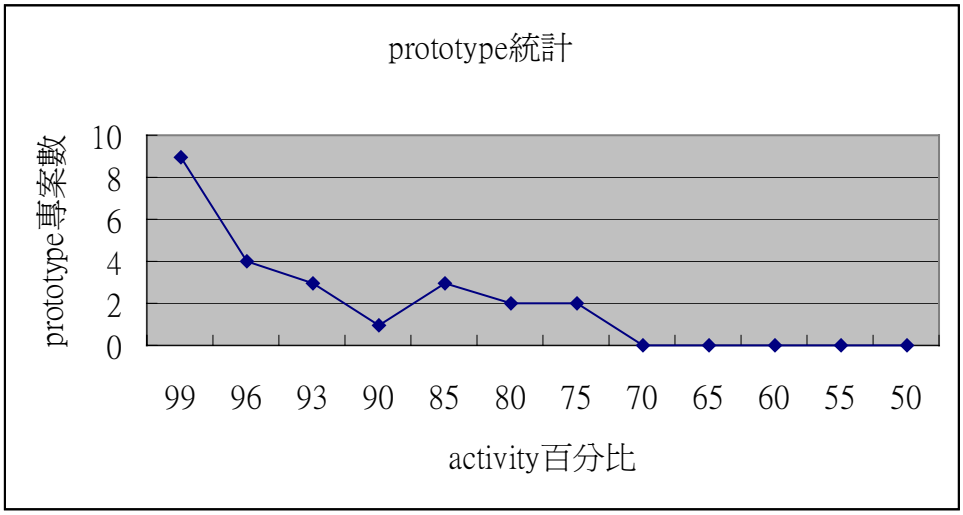


Figure 22 依照活躍度統計符合”Prototype”的專案數

由圖中可以大概看出，活躍度在排行榜中間的專案，受到 prototype 的影響不大，但是在活躍度超過 99%的專案，符合 prototype 的高達 9 個。而活躍度低於 70%以下的 50 個專案，無一符合 prototype。

整體趨勢來看，前兩個月有超過 50 個 CVS comments 的專案，活躍度表現會較佳。尤其是活躍度超過 99%的專案，平均前兩個月都有超過 300 個以上的 CVS comments。由此可以看出，用 prototype 的方式來預期專案會有較佳表現的方式，會比單純期待年紀大的專案好得多。有些專案是一開始就註冊但無任何進展，所以無法吸引使用者的注意。一開始就有雛型的專案則不同，它不必讓使用者等專案註冊一段時間之後才有版本可以讓使用者使用，而是一開始就有了基本的使用者在加以推廣，所以會在活躍度排行榜上有較佳的表現。

6 結論與未來的工作方向

藉由檢視當月最佳專案的活躍值曲線，除了兩個專案在六個月中活躍值過低(實際上其活躍度值還可以維持在排行榜 90%以上)外，其他 17 個專案都符合持續度(sustainability)的定義。用相同方式去檢視在活躍度排行榜不同區段的專案，可以檢視出在排行榜不同區段，專案活躍值的大略值。透過活躍值與活躍度排行榜的百分比，更能比較出開放程式碼中的 Pareto Distribution。

貢獻度分布方面，可以觀察到發展者貢獻比例仍超過 77%(此處的發展者只包括專案首頁所列的資料)，而貢獻者在核心與使用者介面的貢獻比例約爲 5 比 3。使用者的貢獻佔了 22%左右，與預期不同的是使用者在核心與使用者介面的貢獻比例約爲 2 比 1。不管是發展者或者使用者在核心的貢獻都比使用者介面多，與傳統核心與使用者介面比例不同。可能是擷取專案數太少，樣本數不足，或只計貢獻筆數，而沒有考慮到每筆貢獻重要程度不同或加入的程式碼行數差異。

雛型系統(Prototype)方面，可以觀察出一開始有雛型的專案，在活躍度排行榜上會優於沒有雛形的專案，尤其在排行榜 99%以上的差異更大。若由排行榜爲主體去檢視符合雛型定義的專案，則有侷限性。較合理的方式是統計符合雛型定義的所有專案，觀察排行榜區段分佈數量最多且符合定義的專案，更能看出符合雛形定義專案呈現的活躍度。在排行低於 50%的專案，通常 CVS comments 數不到 50，可以預期的是符合雛型定義的專案幾乎都至少會在排行榜 50%以上。有問題的部分是定義是否恰當，再未來做實驗時，亦可變更標準—前兩個月所需的 cvs comments 的數量，可將 50 改成 30 或 70，看看符合跟不符合雛型定義專案的差異度，也可藉此反推得到一開始具雛形的專案前兩個月合理的 cvs comments 數量須達到多少以上。

我們主要的貢獻在於確認開放程式碼中的高度偏移現象(high-skewed)，以及

統計專案中貢獻度分布，並確認曾當選當月指標專案者是否都維持一定時間的活躍值。

在擷取 SourceForge 上面的專案資料時需特別注意，有些專案的首頁可能未使用到 SourceForge 所提供的空間。例如 OpenWebMail 只用 SourceForge 的 CVS，而沒有利用下載的功能，該專案註冊至今下載數都是 0，也未提供任何 release。而 P2P 軟體中的 Emule，下載數長時期高居第一名，Bug 次數在 SourceForge 上面卻仍是 0。但不顯示 OpenWebMail 沒有 release，而 Emule 也沒有 Bug，仍需查看其他資訊來源。

後續可進行的工作，包括可藉由普遍度/生命力(popularity/vitality)的比值(參照看 4.2 節)，觀察不同軟體類型的比值是否會呈現差異。對於社群叢集大小的計算方面，我們可透過 4.4 節的方法統計現今 SourceForge 上面的社群狀況，再與[18]中的結論進行比較，判斷社群發展狀況是否已達到成熟的階段，藉此統計出來的社群，使貢獻度分布的分析更加準確(原本視爲使用者的帳號可能與該專案發展者隸屬於同一個社群)。同時對於 SourceForge 上面的錯誤(Bug)報告，可藉由 resolution_id 值或檢驗狀態有修改過的報告，統計出有效的比例。

可針對普遍度與生命力相差極大的專案進行探討，歸納出不同類型專案，所具有的普遍度與生命力型態。在 SourceForge 上，P2P 軟體是屬於普遍度(下載數)高但生命力不高者，而 OpenWebMail 由於沒有開放下載，因此普遍度不高但生命力高。這些比較特殊的狀況都可深入討論，也可應用到所提到的水平(horizontal)與垂直(vertical)應用程式(application) 。水平的應用程式代表使用者會利用該程式來開發其他的軟體，例如系統開發軟體、軟體開發環境以及資料庫開發軟體等。而垂直的應用軟體如普遍使用者應用的軟體，可以嘗試歸納兩者普遍度與生命力的差異。

參考文獻

[1] Gregor J. Rothfuss , "A Framework for Open Source Projects" , Master Thesis in Computer Science of Zurich , November 12, 2002

[2] Bill Curtis , William E.Hefley ,Sally A. Miller , "People Capability Maturity Model® (P－CMM®) Version 2.0 ",Software Engineering Institute ,2001 Available WWW：http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01mm001.pdf

[3] Juric, R. , "Extreme programming and its development practices" , Information Technology Inerfaces , 2000. Proceedings of the 22nd International Conference on , June 13-16 , Page(s): 97－104

[4] Laurie Williams , "The XP Programmer:The Few-Minutes Programmer" , IEEE Software,May/June 2003 (Vol. 20, No.3)Available WWW：http://collaboration.csc.ncsu.edu/laurie/Papers/fewMinutes.pdf

[5] Mark C.Paulk , "Extreme Programming from a CMM Perspective" , IEEE Software , November/December 2001 (page 19-26)

[6] Rierson, L.K. , "Using the software capability maturity model for certification projects" , Digital Avionics Systems Conference, 1998. Proceedings. , 17th DASC. The AIAA/IEEE/SAE , Volume:1,31 Oct.-7 Nov. 1998 Page(s): C24/1 -C24/8 vol.1

[7] Andrea Capiluppi,Patricia Lago,Maurizio Morisio , ” Characteristics of Open Source Projects” , Proceedings of the Seventh European Conference On Software Maintenance And Reengineering (CSMR’ 03) , Available WWW：http://softeng.polito.it/andrea/publications/csmr2003.pdf

[8] Gregory B. Newby, Jane Greenberg, and Paul Jones , ” Open Source Software Development and Lotka’ s Law: Bibliometric Patterns in Programming” , Journal of the American Society for Information Science and Technology-January

2002

[9] Yunwen Ye, Kouichi Kishida , ” Toward an Understanding of the Motivation of Open Source Software Developers” , Proceedings of 2003 International Conference on Software Engineering (ICSE2003), Portland, OR, May 3-10, 2003

[10] Kumiyo Nakakoji,Yasuhiro Yamamoto,Yoshiyuki Nishinaka,Kouichi Kishida,Yunwen Ye , ” Evolution Patterns of Open-Source Software Systems and Communities” , International Workshop on Principles of Software Evolution 2002 (IWPSE2002) , Orlando, FL, May 19-20, 2002

[11] Mario A. Maggioni , ” Open Source Software Communities and Industrial Districts: a Useful Comparison?” , Università Cattolica del Sacro Cuore, Milan, Italy February 2002

[12] Audris Mockus , Roy T Fielding , James D Herbslev , ” Two Case Studies of Open Source Software Development: Apache and Mozilla” , ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3, July 2002, Pages 309 – 346.

[13] F. Hunt and P. Johnson , “On the Pareto Distribution of SourceForge Projects” , in C. Gacek and B. Arief (eds.), Proc. Open Source Software Development Workshop, 122-129, Newcastle, UK, February 2002

[14] K. Crowston, H. Annabi, and J. Howison, “Defining open source software project success.” , In Proc. of International Conference on Information Systems (ICIS), 2003.

[15] K. Crowston, H. Annabi, J. Howison, and C. Masano. “Towards a portfolio of FLOSS project sucess measures.” , In ICSE Open Source Workshop, 2004.

[16] K. Crowston and J. Howison. ,” The social structure of open source software development teams.” , In OASIS 2003 Workshop (IFIP 8.2 WG), 2003.

[17] Howison, J. & Crowston, K. (2004), “The perils and pitfalls of mining sourceforge.

“ , Workshop on Mining Software Repositories at the International Conference on Software Enginneering ICSE. Edinburgh, UK May 25

[18] G. Madey, Freeh, V., and Tynan, R. “The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory” , Americas Conference on Information Systems (AMCIS2002) . Dallas, TX, 2002. pp. 1806-1813

[19] Francis Hunt and Paul Johnson. “On the pareto distribution of sourceforge projects.” In Proceedings of the Open Source Software Development Workshop, pages 122-129 , Newcastle, UK, 2002.

[20] “Cooking pot market” : an economic model for the trade in free goods and services on the Internet URL:http://www.firstmonday.dk/issues/issue3_3/ghosh