以 COSMIC-FFP 功能點為基礎之軟體維護專案規模估算工具
A Software Maintenance Project Size Estimation Tool Based On Cosmic Full Function Point

林紀睿 葉道明
國立高雄師範大學 軟體工程學系
CHI-JUI Lin, Dow-Ming Yeh
Email: playkant@yahoo.com.tw, dmyeh@nknu.edu.tw

摘 要

軟體規模估算對於專案規劃初期在評估專案人力、時間以及預算具有良好的參考價值，目前大多數的組織都在關注軟體維護成本的研究，因為軟體維護成本一直在不斷成長，很多軟體公司所花費的軟體預算已達到 80%用於軟體維護，這意味著軟體開發的時間與人力成本大多於維護成本，但是維護專案的功能點計算與新開發專案的功能點計算略有不同，如在分析新開發專案時，除了呼叫內建類別庫以外所有其他的訊息都會累計成功能點數，而分析維護專案時如果訊息是傳給原有系統中已經有的物件類別，而非新增加的類別。其對應之功能點數應不予計算，因此便有需要區分新舊類別之功能來以此區分維護成本與開發成本。本研究針對維護專案提出一個方法來區分新舊版本的功能並且開發出一個輔助工具，藉由 COSMIC-FFP 的功能點分析，透過比對兩個版本之間的差異，再利用迴歸分析求出一單位 Cfsu 等於多少程式碼行數以及誤差值，使我們能更精準的評估維護專案的軟體規模大小，以此來分析維護專案的成本。

關鍵字：軟體規模、功能點分析、反向工程、COSMIC-FFP

一、緒論

軟體規模估算對於專案規劃初期在評估專案人力、時間以及預算具有良好的參考價值，軟體規模估算方法有很多種，著名的如:功能點分析，其在估計商業資訊系統規模方面已經獲得廣大的接受度，目前大多數的組織都在關注軟體維護成本的研究，因為軟體維護成本一直在不斷成長，很多軟體公司所花費的軟體預算已達到 80%用於軟體維護[1]。

隨著開源程式碼社群的增長，開源程式碼被越來越人使用，這些開源程式碼被用來進行軟體的二次開發，有許多軟體開發專案便是使用開源程式碼來進行進一步的軟體開發，如今，軟體開發專案大多數是加強或是維護現有的系統，這也意味著當進行軟體開發時，大部分的時間人力成本是用於維護與更新舊有的系統。

一般而言，維護成本費用難以估算，因為這些成本涉及到系統、製程和組織等因素[2]。然而其估算模式與軟體開發成本估算模式類似，幾乎都是以系統規模大小為基礎，如:COCOMO II[3] ，功能點分析[4]是由 Albrecht 在 1979 年所提出的一種軟體規模評估的方法，隨後其他以不同功能概念的功能點分析也應運而生，如: IFPUG[5]、COSMIC-FFP，其中 COSMIC-FFP 已經被越來越多人利用來評估新的專案，因為對於功能點分析來說，COSMIC-FFP 已經被證實它是一個很好的選擇[6]。

可是雖然計算維護專案新增功能的功能點與計算新開發專案之功能點類似，但在分析新開發專案時，除了呼叫內建類別庫以外所有其他的訊息都會累計成功能點數，而分析維護專案時如果訊息是傳給原有系統中已經有的物件類別，而非新增加的類別。其對應之功能點數應不予計算，因此我們須區分新舊類別之功能來以此區分維護成本與開發成本。

本研究針對維護專案提出一個方法來區分新舊版本的功能並且開發出一個輔助工具，藉由 COSMIC-FFP 的功能點分析，透過比對兩個版本之間的差異，再利用迴歸分析求出一單位 Cfsu 等於多少程式碼行數以及誤差值，使我們能更精準的評估維護專案的軟體規模大小，以此來分析維護專案的成本。

二、相關研究

軟體規模的估算時常依賴資深工程師根據類似或已完成的專案經驗為基礎來估算新專案，這種缺乏數據或依靠經驗甚至直覺的估算方式，常會造成軟體規模不當估算的結果。假使低估了軟體規模，不但會引發專案各階段工作無法如期完成，造成專案發展過程的混亂，還會影響專案的品質，而如果高估了軟體規模，則會導致時間和預算的誤判甚至影響其他專案的成本，在對於專案管理來說是一種浪費。因此，軟體規模評估是軟體開發工作中一個重要的項目，良好的規模有助於軟體專案規劃與管理。

學術界與業界定義了若干標準以估計的軟體系統的規模，其中功能點分析(Function Points Analysis, FPA) 在估計商業資訊系統規模方面已取得廣大的接受度，並且能間接地預測其專案的時間、人力成本和持續時間[7]。此方法評估欲開發系統之功能來提供軟體規模之估計值，因此才能在生命周期的初期階段實施，也是此法成功的主要原

因。

Albrecht 的功能點分析(FPA, Function point analysis)被廣泛運用在測量軟體規模大小然而，功能點分析被質疑沒有以客觀的角度去考慮複雜性。因此，FPA 不能適用於所有類型的軟體[8]。這導致了之後有各種以不同功能概念為基礎的功能點分析相繼於不同年間發佈，如:IFPUG FPA[1，5]、Mark-II FPA[1] 、COSMIC-FFP[1,9,20]等，有些方法已經做了一些嘗試去讓 FPA 去適應軟體類型，這些嘗試希望可以更客觀的評估軟體的複雜度,如即時系統,而 Mark-II FPA 或是 COSMIC-FFP 便是這樣的方法之一，IFPUG FPA 則為延續 Albrecht 所提之方法[10]。

IFPUG FPA 是一個長壽的軟體評估方法，他雖然已經被發展出來整整三十年,但依舊被廣泛使用[18]，但是 IFPUG 使用的是名目尺度(nominal scale)，所以 IFPUG 很少、甚至不會在不同專案之間進行比較,也就是說 IFPUG 所求出的 FP(function points)較難與其他專案的 FP 進行比較，如:20FP 並不能說二倍小於或複雜於另一個專案的 40FP，而 COSMIC-FFP 所測量出來的功能點是比例尺度[19],也就是說 20Cfsu 可以說是二倍小於或複雜於 40Cfsu，因為這個原因，我們選擇 COSMIC-FFP 來評估維護專案,我們將評估不同版本的維護專案來進行比較。

COSMIC-FFP 是一種功能點分析的方法，它是由同名組織 COSMIC(Common Software Measurement International Consortium)所提出的軟體規模評估方法,COSMIC-FFP 在進行軟體規模估算的時候，系統功能是以功能流程(process)形式為模型，接著再分成子流程(sub-processes)。子流程分為資料移動(data movement)與資料操作(data manipulation)，主要計算對象為資料移動可區分為四類 ：

- Entry -從使用者到功能流程之資料移動
- Exit -從功能流程到使用者之資料移動
- Read -從持久性儲存（persistent storage）到功能流程之資料移動，持久性儲存需為系統之一部份。
- Write -從功能流程到持久性儲存之資料移動，持久性儲存需為系統之一部份。

功能流程的資料操作通常被假定為包含在資料移動中，不單獨測量，模型如圖 1 所示。

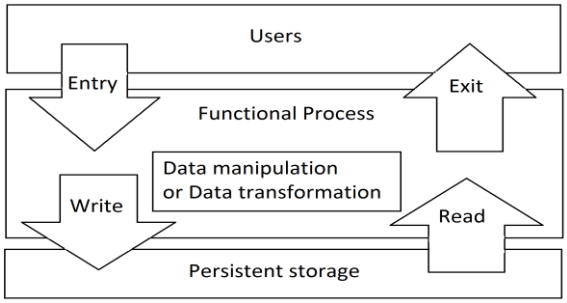

圖 1 COSMIC-FFP (參考[8]所畫)

在某功能流程中，計算上述子流程之 data groups 移動數量即為該功能之點數(單位為 Cfsu – COSMIC functional size unit)。

至於如何使用 COSMIC-FFP 來估算維護專案規模大小，當然還是回歸功能層面。從功能上看，軟體維護專案可能包括四個部分:

- 增加新的功能
- 改變現有的功能
- 刪除現有過時的功能
- 替換現有的功能（=刪除+新增）

因此估算維護專案中受影響的功能點大小就是分別對新增的、修改的、刪除的和替換的功能點進行估算。新增的功能，估算方式與標準的 COSMIC-FFP 方法一樣；對於更改的功能，只測量改變的資料移動數量,COSMIC-FFP 指引對商業應用軟體中功能點改變的定義為當一資料移動相關的 data group 或是資料操作有任何改變；刪除的功能點規模將由所有被刪除的資料移動的總數來決定；更換的功能部分，可以只計算新增功能的資料移動。

這四個部分所評估的功能點點數不完全相等,針對刪除的部分所求出的功能點點數勢必比新增的部分少，Sogeti 建議其權重可設為 0.1[11](即刪除十個功能所評估出來的功能點數等於新增一個功能的功能點數)，但在我們的系統中，評估刪除現有過時的功能總數在測試系統所佔的比例不高，並且對於功能點評估的影響較小，所以在本論文中不予考慮,而替換現有的功能這部份則完全視同為增加新功能的部分。

三、系統設計

我們選用 Python 做為系統的開發語言，因為其中的正則表示式對於程式碼的搜尋十分便利,而測試之程式則選用 Java 之開放程式碼做為測試對象，維護專案則從 SourceForge[12] 獲取，SourceForge 是一個開放式的開源程式碼網站，截止 2015 年,網站上以 Java 專案而言便有 53000 個,因此我們便從其中獲取具有多個版本的維護專案來做為測試對象，測試對象名稱為 Hibernate Search，Hibernate 是在 Java 語言下的物件關係映射框架(object-relational mapping framework)[13]，而 Hibernate Search 則是 Hibernate 的搜尋引擎[13]，Hibernate Search 使用了 Java 的多種語法，如:interface、native 等，因此我們選擇其當作我們的測試對象 。

根據不同功能的定義，我們的系統包含了需求分析、區分管理、功能點分析。系統的基礎架構如圖 2 所示。

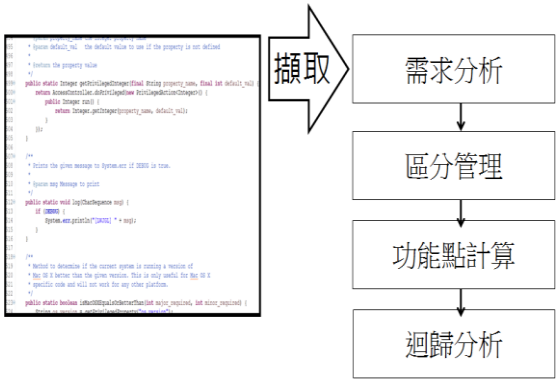

圖 2 系統基礎架構圖

3.1 需求分析

COSMIC-FFP 所使用之需求分析產出物，大多為使用個案和 UML 循序圖(sequence diagram)或類別圖。此章節的目的在於建立維護專案之循序圖資訊，並將此循序圖之資料儲存以便分析階段使用，要精細地建構一現存物件導向系統之循序圖(sequence diagram)，需要此系統之架構與行為方面的資訊。若無完整且一致的需求設計模型能提供這些資訊，我們只能求助於反向工程(reverse engineering)，藉由種種的靜態和動態分析盡可能取得所有可得之資訊。過程以下述程式碼說明:

```
class A{...}
class B{
public void func1(A a){
        a.mes1();
        A b = this.func2(a,5);
        b.mes6();
        }
public void func2(A a2,int n){
        a2.mes2();
        if(n>0){A d = this.func3(a2); d.mes5();}
        return
        }
public void func3(A a3){
        a3.mes3();
        A e = this.func4(a3);
        return a3;
        }
public void func4(A a4){
        a4.mes4();
        return a4;
        }
}
```

產生之循序圖如圖 3

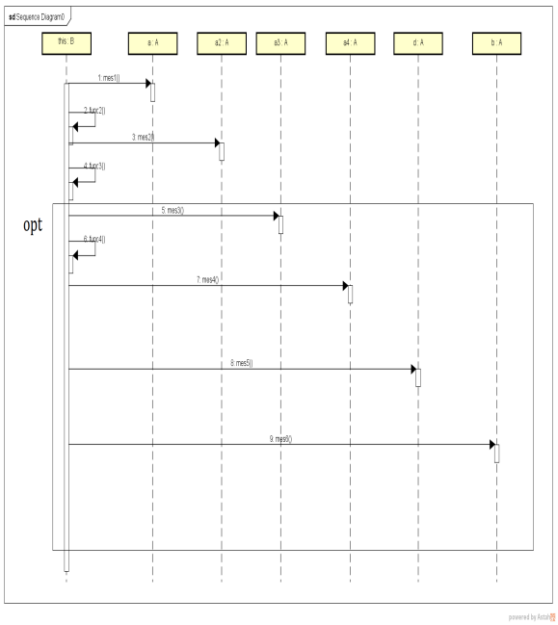

圖 3 循序圖

上圖所繪製的循序圖並不完全正確，a、a2 可以看成同一物件，但是以功能點分析的角度來看，合併來看並不會影響評估結果，因為我們在此系統是計算圖中訊息傳遞的數量，並不再對此進行整合的處理。

從程式碼以及圖 3 可以看出 class A 所涉及到的功能以及功能所具有的一些屬性，如:method 名稱、呼叫指令，這些屬性我們這類的屬性通稱功能屬性，依照資料移動所涉及到的功能，也就是功能流程，我們會抓取維護專案裡所有的功能流程以及其中每一個功能的屬性，並且擷取功能所在的位址，最後再將上述的功能屬性、位址儲存以便分析階段使用。

3.2 區分管理

如第一章節提到的，為了分析並評估維護專案的規模大小，我們須區分新舊類別之功能，如果訊息是傳給原有系統中已經有的物件類別，而非新增加的類別。其對應之功能點數應不予計算，因此在進行功能點分析之前，我們必須先針對維護專案的版本之間進行比對，以此來區分新版本和舊版本之間的差異。

在此必須提醒的是，在我們的系統中，我們評估的對象為維護專案，即測試對象具有兩個版本以上的專案。步驟如下所述:

步驟一，我們會抓取上一個版本以及新的版本這兩個維護專案的功能屬性資料，其資料經過 3.1 章節所描述的需求分析。

步驟二，依照功能屬性裡的位址去抓取兩個版本的程式碼，從前後這兩個版本的程式碼之中，依照新的版本的 method 名稱去抓取兩者程式碼 method 的程式碼，如圖 4，去除其中的空白字符降低不必要的干擾，然後分批存入兩個不同的 list 中。

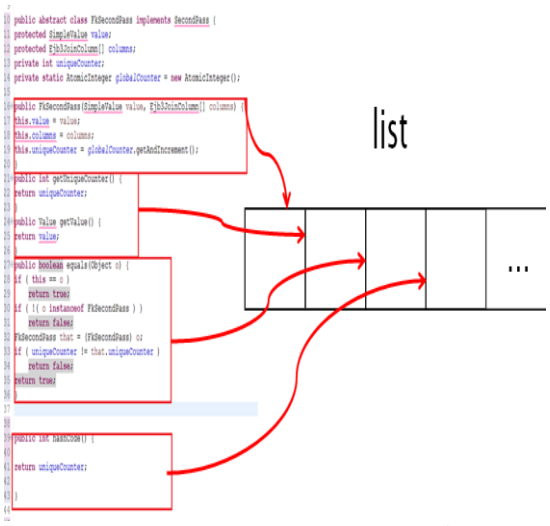

圖 4 擷取 method 的程式碼並存取

步驟三，擷取兩個版本的 method 程式碼，由新的版本去比對上一版本的程式碼，如果比對結果不一樣，則認定此 method 為更新或維護之程式碼，予以計算其功能之點數，如果比對結果為一樣，我們會忽略此 method 的 data groups，即不計算該功能之點數，在此必須提到，如果上一版本之功能在新的版本被刪除的話，在此步驟是無法被發現的，但是此為刪除過時之功能，對於功能估算影響不大，故此忽略。

3.3 功能點分析

COSMIC-FFP 主要計算對象為資料移動，在我們的系統中，此階段所進行的動作與需求分析階段十分類似，差別在於功能屬性已經在需求分析階段就被擷取出來，並且在分析功能流程時，必須同時計算資料移動，步驟如下所述:

步驟一，首先從 3.1 已經處理好的循序圖層次之資訊擷取相關的功能屬性(如:class 名稱、method 名稱、package 名稱、檔案所在的位址等)，依照擷取出來的功能屬性去搜尋專案所在的位址並且依據功能屬性裡的 class 名稱找到對應的程式碼，最後把此 class 的程式碼存入 list 中，去除其中程式碼的註解以便之後運用。

步驟二，透過處理過的程式碼以及擷取出來的功能屬性，我們可以依據資料移動找到所對應的功能流程(可能是一個或是多個)，依照 3.2 章節所講解的區分管理步驟，對於比對結果一致的資料移動不予計算，而比對結果不一致的資料移動則予以計算，計算資料移動的結果可以為 0，因為所評估的功能可能是不曾更新過或維護過的。

步驟三，功能點計算有可能並非為整數的 Cfsu 單位，其單位有可能為小數點，功能點計算的小數點可能來自多模(polymorphism)或是 interface，單由程式碼分析無法確定是哪一個子類別的方法被執行或是哪個實作介面的方法被執行時，系統會取所有可能方法之平均值加以計算，而對於一個維護專案來說，它可能有一個或是多個功能流程，所以當上述步驟完成之後，加總所有功能流程的功能點總和，便會獲得此為維護專案之功能點。

四、系統展示

我們在第 3 章節已經完整的介紹系統的設計，此章節則會展示系統的相關結果，以及對結果進行迴歸分析。

圖 5 為我們的系統在評估維護專案的畫面，由上而下，針對的部份分別為繼承、多模以及結果，第一個是在判斷是否有繼承存在，第二個則是針對多模或是介面，當它們被執行時，系統會取所有可能方法之平均值加以計算，第三個顯示的是維護專案評估結果，FP 是維護專案比對過後所顯示新版本的評估，比較需要提到的是陣列第一個數字為功能點點數，而第五個為程式碼行數，Old FP 則是顯示兩個版本比對一致的結果，也即是評估舊有版本便擁有之功能。以此知道在維護專案中，新的版本相較於舊有的版本所更新或維護的功能點點數，這將能更好的評估維護專案的規模大小。

最後依照所得到的數據進行迴歸分析，如圖 6 中 D、E 欄，在此顯示的是完成比對的結果，我們依照十個不同功能流程所評估出的功能點進行迴歸分析，即是區分出新舊版本之後的新維護專案，其去除原有系統中已經有的物件類別所計算的功能點數，在 D、E 欄有部分功能流程評估結果的功能點與行數為 0，此代表比對之功能並未進行更新或維護的動作不予考慮，最後得出的結果由迴歸分析得出一單位之 Cosmic-FFP 約等於多少行程式碼與其誤差情形。

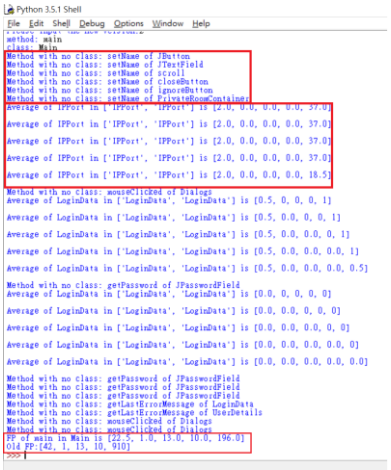

圖 5 估算工具評估結果

五、問題

在此必須要提到的是測試對象為從開源程式碼社群中取得的開放式程式碼，我們無法對程式撰寫者程式撰寫風格進行更進一步的交流，這導致系統在進行分析的時候的判斷困難，例如在 Java 中，interface 是可以存在在 class 裡面與 class 以外的，雖然這對於 Java 在執行的時候並不會帶來太大的影響，但是站在靜態分析的角度上，在 class 裡面

	A	B	C	D	E
1	功能點數(x)	行數(y)	功能之method與class	功能點數(x)	行數(y)
2	1.45	4.7	Truck Truck	1	1
3	1.5	16	loadTestData WebShopTest	1.5	16
4	9.5	54	SearchService SearchService	0	0
5	1	2	isSelected FacetMenuItem	0	0
6	4	15	testNullIndexingWithDSLQueryIgnoringFieldBridge IndexAndQueryNullTest	4	15
7	7	83	testProjectedValueGetsConvertedToNull ProgrammaticIndexAndQueryNullTest	8	90
8	7	59	testOnlyOneCriteriaQueryIsUsedToLoadMatchedEntities ObjectLoaderHelperTe	8.5	67.5
9	1	2	getExploredCountries Explorer	1	2
10	6.833	58.33	insertTestData SortOnFieldsFromCustomBridgeTest	6.833	58.33
11	1.5	13.5	testResultOrderedByIdAsString SortTest	2.75	21.5
12	相關係數	0.89175972		相關係數	0.95388
13	迴歸方程式	y=a+bx		迴歸方程式	y=a+bx
14	斜率(a)	8.2220371		斜率(a)	9.46443
15	截距(b)	-2.7789339		截距(b)	-4.6514

圖 6 迴歸分析比較

或是外面是一個蠻困擾的影響，因為在 Python 的正則表示式中，物件的先後是十分重要的，這甚至有可能導致無法成功擷取正確的資料，而這對於資料處理來說是十分麻煩的，也因此我們的系統在對於 Java 程式語言中的 interface 與 interface 裡面的 method 的判斷是有瓶頸的，雖然曾經一度考慮在評估的工程中忽略 interface，但是 interface 在軟體評估中是有所影響的。圖 6 為我們實際評估有無 interface 之結果，A、B 欄呈現的是忽略計算 interface 的情況，也就是說本研究所開發的輔助工具在評估 Java 維護專案時，忽略 interface 以此來考量是否 interface 對於軟體評估有所影響，D、E 欄則為評估 interface 的結果，比較雙方可以發現在評估時，忽略 interface 將影響功能點數以及行數的計算，除此之外，區分管理也因此而受到影響，在圖 6 的 4、5 行便可以比對出差異，忽略 interface 的輔助工具判斷此兩項功能的功能流程為維護專案，即是具有更新或維護之功能，但是在考量 interface 的結果可以發現這兩項功能並不曾經過更動，也因此可以發現忽略 interface 對於我們的系統在評估維護專案是有影響的。

六、結論與未來計畫

本研究針對維護專案提出一個方法來區分新舊版本的功能並且開發出一個輔助工具，藉由 COSMIC-FFP 的功能點分析，透過比對兩個版本之間的差異，再利用迴歸分析求出一單位 Cfsu 等於多少程式碼行數以及誤差值，此方法可以更明確的評估維護專案的規模大小。

由於維護成本除了涉及到系統，還涉及到製程和組織等因素[2]，物件導向度量可以幫助使用者了解物件導向設計的複雜性並且預測某些專案的成果，除此之外，物件導向系統之複雜度度量對於設計成本有明顯的影響[14]。

最早的物件導向系統度量集之一是由 Chidamber 和 Kemerer 提出，因此他們所提出之度量集通常被稱為 CK 度量[15]，CK 度量指標集及其他配套的量度在業界的接受度越來越高，在學術界大量被引用[16]，這些度量指標在現實世界中的實驗驗證也日漸增加，而 CBO（coupling between object classes）為 CK 度量中的一種指標，Chidamber 和 Kemerer 提出繼承與非繼承耦合之論點，Dagpinar 和 Jahnke 也提出物件導向度量中耦合度與軟體規模大小具有相關性[17]，因此我們希望在未來可以擴展輔助工具並且把耦合度量也考慮進維護專案的軟體評估中，同時也希望導入更多不同的測試案例以此來從不同方面驗證此系統。

參考文獻

[1]Yunsik Ahn*,†, Jungseok Suh, Seungryeol Kim and Hyunsoo Kim. “The software maintenance project effort estimation model based on function points”. JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE J. Softw. Maint. Evol.: Res. Pract. 2003; 15:71–85

[2] Yeh, D., & Jeng, J. H. (2002). An empirical study of the influence of departmentalization and organizational position on software maintenance. Journal of Software Maintenance and Evolution: Research and Practice, 14(1), 65-82.

[3] Boehm, B. 2000. "COCOMO II Model Definition Manual " . Center for software Engineering,University of Southern California.

[4]WIKIPEDIA. (Online). “Function point”.Available: https://en.wikipedia.org/wiki/Function_point

[5]WIKIPEDIA. (Online). “IFPUG”.Available: https://en.wikipedia.org/wiki/IFPUG

[6]Tom Koppenberg, Ton Dekkers. ”Estimating maintenance projects using COSMIC-FFP”. IWSM/MetriKon 2004

[7]P. Nesi and T. Querci, “Effort Estimation and Prediction of Object-Oriented Systems,” J. Systems and Software, vol. 42, pp. 89-102,1998.

[8]De Tran-Cao, Ghislain Levesque , Alain Abran3“Measuring Software Functional Size:Towards an Effective Measurement of Complexity”

[9]Alain Abran, S. Oligny, and Charles R. Symons .“COSMIC Full Function Points (FFP) and the Worldwide Field Trials Strategy.” ,(Online).Available:http://www.ittoday.info/Articles/COSMIC_FPP/COSMIC_FPP.htm
[10]Dumke, R., & Abran, A. (Eds.). COSMIC Function Points: Theory and Advanced Practices. CRC Press.
[11] Engelhart, J.T., Langbroek, P.L., Dekkers, A.J.E., Peters, H.J.G., Reijnders, P.H.J., Function point analysis for software enhancement, A professional guide of the Netherlands Software Metric Users Association, NESMA 2001
[12]SourceForge(2015).[Online].Available:https://sourceforge.net/
[13]HIBERNATE. (Online). Available: http://hibernate.org/
[14] Chidamber, S.R.; Darcy, D.P.; Kemerer, C.F.; , "Managerial use of metrics for object-oriented software: an exploratory analysis,", IEEE Transactions on Software Engineering, vol.24, no.8, pp.629-639, Aug 1998.
[15] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object-Oriented Design,” IEEE Trans. Software Eng., vol. 20, pp. 476-493,1994.
[16] K. El Emam, S. Benlarbi, N. Goel, and S.N. Rai, “The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics,”IEEE Trans. Software Eng., vol. 27, pp. 630-650, 2001.
[17] Dagpinar, M., & Jahnke, J. H. (2003, November). Predicting maintainability with object-oriented metrics-an empirical comparison. In 2003 10th Working Conference on Reverse Engineering (WCRE) (pp. 155-164). IEEE Computer Society.
[18] Md. Forhad Rabbi,Shailendra Natraj, Olorisade Babatunde Kazeem,” EVALUATION OF CONVERTIBILITY ISSUES BETWEEN IFPUG AND COSMIC FUNCTION POINTS”, 2009 Fourth International Conference on Software Engineering Advances
[19]What is the biggest difference between COSMIC and FPA? (Online). Available: http://cosmic-sizing.org/faq/biggest-difference-between-cosmic-and-fpa/
[20] Muhammet Ali SAĞ, Ayça TARHAN,” Measuring COSMIC Software Size from Functional Execution Traces of Java Business Applications”, 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement