Presented at the 2011 ISPA/SCEA Joint Annual Conference and Training Workshop - www.iceaaonline.com

cobec CONSULTING

QUANTIFYING THE FUTURE

2011 SCEA Conference Presentation

Function Point Analysis: One Size Fits All

Dan French, CFPS
dfrench@cobecconsulting.com

Program

- Introduction
- Origins of Function Points
- Common Misconceptions Regarding Function Points
- Challenges of Using SLOC
- Cautionary SLOC Tale # 1- International Olympiad of Informatics
- Cautionary SLOC Tale # 2 - Personal Software Process class
- What are (and aren't) Function Points
- Key Function Point Concepts
- Benefits Function Points Provide
- Conclusion
- Addendum
  - Other Sizing Methods
  - FPA Mind Map
  - Benford's Law Validation of the Methodology
  - Resources

Introduction

- B.S. in Economics
- Over 10 years experience in software cost estimation
- Counting function points for 12 years and been a Certified Function Point Specialist (CFPS) for 10 years
- Experience in a number of estimation techniques and tools including SEER-SEM, COCOMO, SLiM, Delphi, and Estimating by Analogy
- Member of the International Function Point Users Group (IFPUG) New Environments Committee since 2004
- Former member of the IFPUG Conference Committee for 5 years
- GAO Cost Guide expert team member
- Project Management Institute (PMI) Project Management Professional (PMP) certification expected in 2011
- Member of the GAO Cost Guide expert group

The Origins of Function Points

- In the late 1960's IBM identified an issue with the accuracy of SLOC based estimates for software development projects.
- Assigned Allan Albrecht to develop an alternative method to measure software
- Created Function Points as an alternative to Source Lines of Code (SLOC) for measuring software size
- Presented methodology for the first time in 1979 in his paper "Measuring Application Development Productivity"
- First formal counting guidelines published in 1984
- Counting Rules are established by the International Function Point Users Group (IFPUG), founded in 1986
- Current version is 4.3.1, Released in January 2010
- International Standards Organization (ISO) Standard for software functional sizing (ISO/IEC 20926 Software Engineering - Function Point Counting Practices Manual)

Misconceptions Regarding Function Points

- Can't be applied to some applications, languages, or platforms
- Can't be applied to real-time or embedded systems
- Is too difficult to learn and understand
- Is too time consuming
- Too costly
- There are tools that can automatically count Function Points

Challenges of Software Lines of Code

- No defined counting rules or standards organization
- Language and platform dependent
- Inconsistent rules mean there is no reliable and verifiable industry data
- Penalizes efficient software writing, incentivizes poor coding
- Heavily dependent upon developer skill and style
- Difficult to estimate early in lifecycle

Function Points address and overcome these challenges

SLOC Cautionary Tale #1-IOI

The International Olympiad in Informatics is an annual gathering of the top software coders in the world under the age of 18. It is a two day individual competition to develop software programs to solve 8 problems.

In 2010, the two favorites to win were Gennady Korotkevich from Belarus and Neal Wu from the USA.

Here is an example of how each solved one of the problems.[1]

The Challenge

- Must solve a murder mystery similar to the game "CLUE"
- The program must determine the murderer, crime scene, and weapon
- There are six possible murderers (1-6), 10 locations (1-10), and six weapons (1-6).
- The Detective tries to guess the correct combination (there are 360 possibilities) via queries to the assistant
- The assistant confirms or refutes each guess and reports which one of the guesses—murderer, location, or weapon—is wrong.

Korotkevich's Solution

Language: Pascal
Lines of code: 22

```
unit cluedo;

interface
   procedure Solve;
implementation
   uses graderlib;
   procedure Solve;
   var
     x,y,z,t: longint;
   begin
     x:=1; y:=1; z:=1;
     t:=theory(x,y,z);
     while t <> 0 do
     begin
       if t = 1 then inc(x) else
       if t = 2 then inc(y)
       else inc(z);
       t:=theory(x,y,z);
     end;
   end;
begin
end.
```

Wu's Solution

Language: C++
Lines of code: 46

```
#include "grader.h"
#include "cluedo.h"

const static int M = 6, L = 10, W = 6;

bool mur [M], loc [L], wep [W];

void Solve ()
{
    for (int i = 0; i < M; i++)
        mur [i] = true;

    for (int i = 0; i < L; i++)
        loc [i] = true;

    for (int i = 0; i < W; i++)
        wep [i] = true;

    int result = -1;

    while (result != 0)
    {
        int m = -1, l = -1, w = -1;

        for (int i = 0; i < M; i++)
            if (mur [i])
                m = i;

        for (int i = 0; i < L; i++)
            if (loc [i])
                l = i;

        for (int i = 0; i < W; i++)
            if (wep [i])
                w = i;

        result = Theory (m + 1, l + 1, w + 1);

        if (result == 1)
            mur [m] = false;
        else if (result == 2)
            loc [l] = false;
        else if (result == 3)
            wep [w] = false;
    }
}
```

SLOC Cautionary Tale #2-PSP Class

- A series of Personal Software Process^{SM} (PSP) courses provided the source of the data
- The requirements, instructor, and the lines-of-code counting specifications for these programs were the same
- A total of 60 sets of nine programs from 5 classes
- Rigorous and consistent collection of measurement data prescribed as part of the PSP (and in the classes)
- Software programs written from the same requirement set, validated by the same instructor, using the same language, and counted the same way.

Findings

- The percentage of variation between the least amount of SLOC for each of the nine programs ranged from a low of 150% to a high of 467% for one language
- In one class of 10, the range of variance for each of the 9 programs was 381% to 2223%
- Within one class, there was considerable variation by program between who coded the least and most with 5 out of the 8 coding both the least and the most SLOC for at least one program
- This was a rare opportunity to confirm, in a controlled environment, the unreliability of using SLOC to size software

What are (and aren't) Function Points?

- Function Points are a standard unit of software size measure
- A reliable measure of the work product of software development
- Measured in terms of functionality from the user perspective
- Functions points **do not** measure internal architecture or technological complexity of an application (how hard it is to build)
- They do not measure effort or duration directly
- Cannot be used to measure individual productivity
- Do not measure technical requirements
- They are not a silver bullet

Function Points Key Concepts

- **User** - any person or thing that communicates or interacts with the software at any time
- **Boundary** - conceptual interface between the software under study and its users
- **Elementary Process** - Smallest unit of meaningful activity to the user

Function Points Transaction Types

- Five Functional Components, 3 Transactional and 2 Data
  - Transaction Functions
    - External Inputs (EI) – Batch transaction file, input screen, control information
    - External Outputs (EO) – Reports with calculations, output files with derived data
    - External Inquiries (EQ) – On-line query screen, interface file with no calculations or derived data
  - Data Functions
    - Internal Logical Files (ILF) – Application file, internal database
    - External Interface File (EIF) – Reference

Steps in FP Counting Process

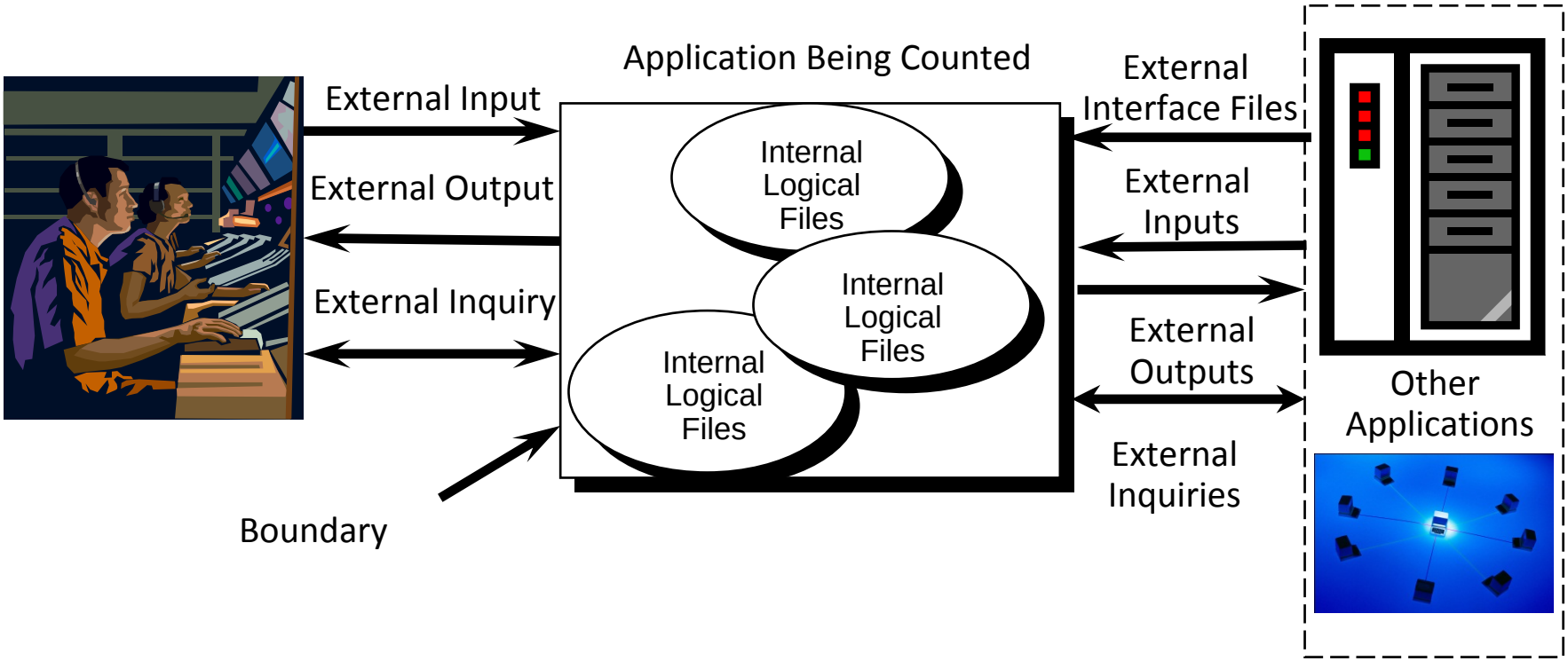
- Determine Type of Count (Development, Application, or Enhancement)
- Identify Counting Scope and Application Boundary
- Count Data Functions
- Count Transactional Functions
- Determine Unadjusted Function Point Count
- Determine Value Adjustment Factor*
- Calculate Adjusted Function Point Count*

* Not part of the ISO standard, but are part of the IFPUG FP Methodology

Function Point Calculation

- Each of the five functional components are evaluated utilizing a matrix which determines the transactions complexity (Low, Average, High)
- Complexity of each transaction has a corresponding function point value
- Summing all the transaction function points represents the functional size of the project or application under analysis

Function Point Conceptual Diagram



Functionality as viewed from the user's perspective

Benefits Function Points Provide

- Improved software project estimation
- Better understanding of enhancement and new development productivity
- Better understanding of application maintenance productivity
- More effective management of project scope and requirements changes
- Improve communication between developers and users
- Facilitation of gathering and refining user requirements
- Development and documentation of a software functionality portfolio

Conclusion: Why Function Points Are One Size Fits All

- Oldest and most utilized functional size metric
- Consistent set of rules
- Platform and language independent
- Functional vs. technical viewpoint
- Can be applied to all software applications
- More accurate estimation
- Better metrics

Questions?

Addendum

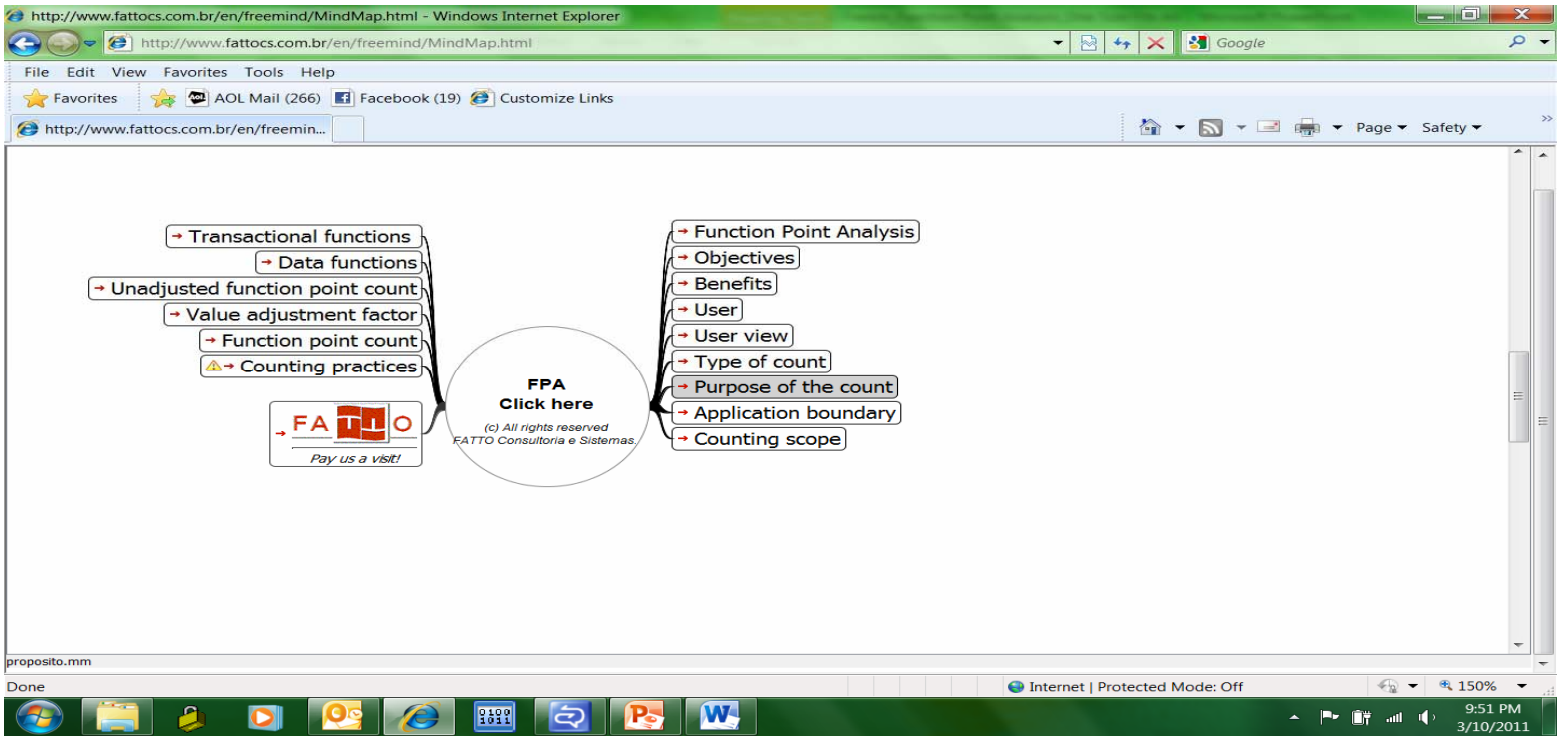
- Some additional information regarding function points, software sizing, and software estimation

Other Sizing Methodologies

- Cosmic Full Function Points
- Mark II (Mk II) Function Point
- NESMA Function Points
- Web Points
- Objects/Function Based Sizing
- Fast Function Points
- Software Non-functional Assessment Process (SNAP)

Function Point Mind Map

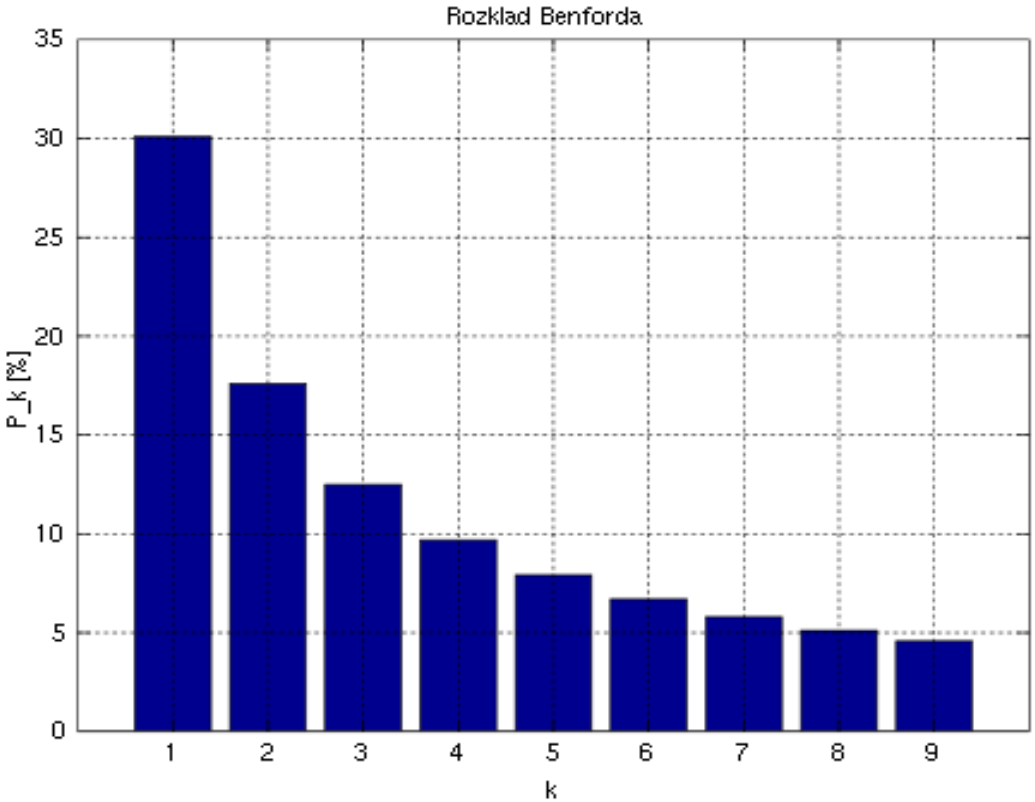
- http://www.fattocs.com.br/en/freemind/MindMap.html



Benford's Law Validates the FPA Methodology

- **Benford's law**, also called the **first-digit law**, states that in lists of numbers from many (but not all) real-life sources of data, the leading digit is distributed in a specific, non-uniform way. According to this law, the first digit is 1 about 30% of the time, and larger digits occur as the leading digit with lower and lower frequency, to the point where 9 as a first digit occurs less than 5% of the time.

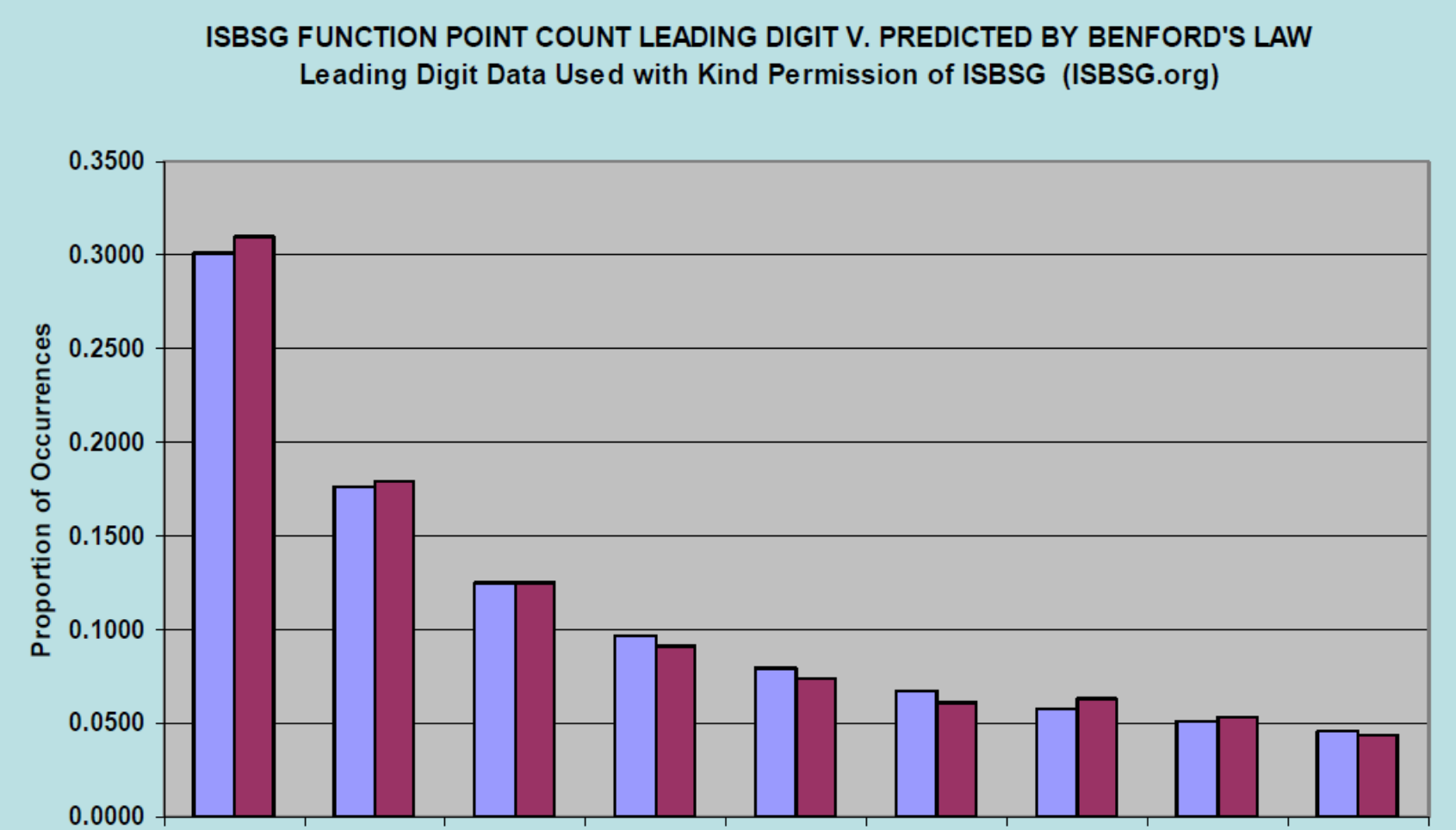
Benford's Distribution



Dr. Charley Tichenor's Hypothesis

- At the 2009 IFPUG Annual Conference, Dr. Tichenor presented his paper "Why Function Point Counts Comply with Benford's Law"
- His hypothesis: Is software development a stimulus and response activity?
- If so, then the stimulus might be the amount of functionality delivered to the user. If this is true, then we can measure the amount of stimulus as the function point count.
- Function point counts should follow Benford's Law.

Charley's Findings



Resources

- International Function Point User Group (IFPUG) http://ifpug.org/
- International Software Benchmarking Standards Group (ISBGS) http://www.isbsg.org
- CrossTalk Magazine http://www.crosstalkonline.org/
- COSMIC http://www.cosmicon.com/
- NESMA http://www.nesma.nl/section/home/
- Systems and Software Consortium http://www.software.org/