Function Point Analysis – A Cornerstone to Estimating

September, 2010

ISMA Cinco!

São Paulo, Brazil

Joe Schofield
joescho@joejr.com
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

A Quick Look Back & Updates on Recent IFPUG / ISMA Presentations

2009	*Counting Lines of Code: Virtually Worthless for Estimating and Software Sizing*, IT Metrics and Productivity Journal; December, 2009 *Is There a Weakest Link After All?,* IT Metrics and Productivity Journal; December, 2009 *Is There Value to using Lines of Code for Measuring People After All?,* IT Metrics and Productivity Journal; December, 2009 *Leaning Lean Six Sigma for Results;* ISMA; September, 2009 *When Did Six Sigma Stop Being a Statistical Measure?; CrossTalk, April 2006* *Lean Six Sigma - Real Stories from Real Practitioners*; Albuquerque, N.M.; N.M. SPIN; August 2005 *Six Sigma & Software Engineering: Complement or Collision*; Albuquerque, N.M.; N.M. SPIN; August, 2004
2008	*Estimating Latent Defects Using Capture-Recapture: Lessons from Biology; Arlington, VA.; 2008 International Software Measurement and Analysis (ISMA) Conference; September 18, 2008* *Beyond Defect Removal: Latent Defect Estimation with Capture Recapture Method*; CrossTalk, August 2007 (reprinted in IFPUG's MetricViews, Winter 2008) *Latent Defect Estimation - Maturing Beyond Defect Removal using Capture-Recapture Method*; QAI QAAM Conference; September 10, 2008
2007	*'Manda, Panda, and the CMMI(R)*; Las Vegas, NV.; 2007; ISMA Conference; September 14, 2007
2006	*Defect Collection & Analysis – The Basis of Software Quality Improvement*; ISMA Conference, September, 2006 *Defect Management through the Personal Software Process℠; CrossTalk, September 2003* *The Team Software Process℠ – Experiences from the Front Line*; Software Quality Forum; Arlington, Virginia, March; 2003 *Measuring Software Process Improvement - How to Avoid the Orange Barrels*; System Development, December 2001 *Usable Metrics for Software Improvement within the CMM*; Software Quality Forum 2000; Santa Fe, N.M.; April, 2000
2004	*Applying Lean Six Sigma to Software Engineering;* IFPUG Conference; September, 2004
2003	*Amplified Lessons from the Ant Hill – What Ants and Software Engineers Have in Common*; IFPUG Conference, Sept., 2003 *Lessons from the Ant Hill - What Ants and Software Engineers Have in Common*; Information Systems Management, Winter 2003
2002	*Lines of Code - Statistically Unreliable for Software Sizing*?; Computer Aid, Inc.; Webinar; October 14, 2008 *The Statistical Case Against the Case for using Lines of Code in Software Estimation;* 4th World Congress on Software Quality; Bethesda, MD.; September 17, 2008 *The Statistically Unreliable Nature of Lines of Code;* CrossTalk, April 2005 (Reprinted at least twice, cited by NIST *Metrics and Measures* http://samate.nist.gov/index.php/Metrics_and_Measures) *A Practical, Statistical, and Criminal Look at the Use of Lines of Code as a Software Sizing Measure*; N.M. SPIN; March, 2004 *Counting KLOCs – Software Measurement's Ultimate Futility (I can't do this anymore, or who am I fooling?, or why not count ants?);* IFPUG Conference; September, 2002

Quick Overview . . .

- We still need improvement in sizing software products and planning software projects
- How Function Point Analysis addresses significant aspects of this need
- Lines of code as a sizing measure has limited potential to address this need
- Estimating with Function Points, from planning to deployment (and beyond)
- Five variance scenarios using a house instead of software as an example
- Mitigating the sources of variance in estimating and performance – an actual exercise
- Models and prediction
- Latent defects (a statistical prediction technique)
- Summary of thoughts presented

Why Projects Stumble

Standish Chaos Report
Challenged projects suffer from:

1. Lack of User Input
2. Incomplete Requirements and Specifications
3. Changing Requirements and Specifications
4. Lack of Executive Support
5. Technology Incompetence (DTRA, XML?)
6. Lack of Resources
7. Unrealistic Expectations
8. Unclear Objectives
9. Unrealistic Time Frames
10. New Technology

Impaired (cancelled) projects suffer from:

1. Incomplete Requirements
2. Lack of User Involvement
3. Lack of Resources
4. Unrealistic Expectations
5. Lack of Executive Support
6. Changing Requirements and Specifications
7. Lack of Planning
8. Didn't Need it Any Longer
9. Lack of IT Management
10. Technology Illiteracy

IEEE Spectrum, Robert N. Charette, September, 2005
Why Software Fails

1. Unrealistic or unarticulated project goals
2. Inaccurate estimates of needed resources
3. Badly defined system requirements
4. Poor reporting of the project's status
5. Unmanaged risk
6. Poor communication among customer, developers, and users
7. Use of immature technology
8. Inability to handle the project's complexity
9. Sloppy development practices
10. Poor project management

How Function Point Analysis Helps . . .

- As an ISO standard (ISO/IEC 20926 SOFTWARE ENGINEERING) Function Point Analysis (FPA) provides a basis for repeatable and consistent sizing
- Supported by IFPUG and its membership community, FPA remains viable as new technologies and approaches to software development evolve (case studies, books, conferences, workshops, certifications, and, the "standard")
- Functional sizing is not influenced by programming language, in-house or COTS development
- Functional sizing is not impacted by development approach: outsourcing, insourcing, iterative, incremental, scrum, or agility
- Functional sizing can be approximated at the first sighting of customer requirements, estimated with a design, and counted upon delivery
- FPA can be used to track requirements volatility over the life of a project (FPs added, changed, deleted) to size *requirements creep*

Examples of the diverse usage of FPA
(from Capers Jones)

Products	Circa 2009 Available	Circa 2018 Available	Daily usage (hours)
Home computer	1,000,000	2,000,000	2.5
Automobile	300,000	750,000	3.0
Smart appliances	100,000	750,000	24.0
Televisions	25,000	125,000	4.0
Home alarms	5,000	15,000	24.0
Home music	7,500	20,000	2.5
I-Phone	20,000	30,000	3.0
Digital camera	2,000	5,000	0.5
Electronic books	10,000	20,000	2.5
Social networks	25,000	75,000	2.5
TOTAL	1,494,500	3,790,000	20.5

Using Function Point Metrics For Software Economic Studies, Capers Jones, January 2010

Lines of code (LOCs) and backfiring can't work

Lines of code **"This term is highly ambiguous and is used for many different counting conventions. The most common variance concerns whether physical lines of logical statements comprise the basic elements of the metrics. Note that for some modern programming languages that use button controls, neither physical lines nor logical statements are relevant."** *Software Quality*, Capers Jones, International Thomson Computer Press 1997, pg. 333.

Direct conversion from source code volumes to an equivalent count of function points is termed *backfiring*. Although the accuracy of backfiring is not great, because individual programming styles can cause wide variation in source code counts, it is easy and popular. *Estimating Software Costs*, Capers Jones, McGraw-Hill, 1998, pg. 191

Software Sizing Problems. "14. Validating or challenging the rules for *backfiring* lines of code to function points." *Estimating Software Costs*, . . . , pg. 322

Of software projects measured in 2001 *backfiring* was used the most for determining size of product, some 75,000 times. LOCs were used 25,000 times. (130,000 projects in survey) *Software Measurement and Metrics: The State of the Art in 2001*; Capers Jones, Software Productivity Research, Inc., October 2001

Actual Data from Three Languages

Course	Attendee	*P1	P2	P3	P4	P5	P6	P7	P8	P9
1	Attendee 1	89	34	67	40	102	235	23	38	168
1	Attendee 3	82	23	33	48	61	34	33	27	52
1	Attendee 4	177	119	67	85	136	276	165	112	233
1	Attendee 5	76	48	305	244	61	121	66	77	127
1	Attendee 7	46	33	17	37	60	95	129	46	186
3	Attendee 5	22	40	100	58	68	131	58	58	102
3	Attendee 6	46	20	30	42	73	82	51	72	82
2	Attendee 7	95	155	147	94	54	191	174	102	218
	Min	22	20	17	37	54	34	23	27	52
	Max	177	155	305	244	136	276	174	112	233
	% *Variation*	805	775	1794	659	252	812	757	415	448
	Mean	79	59	96	81	77	146	87	67	146
	Std. Dev.	47	50	95	69	28	82	60	30	66
	Attendee	*P1	P2	P3	P4	P5	Attendee 9	64	63	36
2	Attendee 5	193	137	48	102	107	Attendee 10	101	116	108
1	Attendee 2	77	163	168	123	134	*Min*	35	30	15
3	Attendee 1	73	37	36	95	101	*Max*	221	311	271
3	Attendee 2	74	97	143	153	279	% *Variation*	631	1037	1807
3	Attendee 4	114	71	108	80	219	*Mean*	113	121	89
							Std. Dev.	56	81	73
	Min	73	37	36	80	101	138	51	66	134
	Max	193	163	168	153	279	207	238	178	305
	% *Variation*	264	441	467	191	276	150	467	270	228
	Mean	106	101	101	111	168	169	145	97	184
	Std. Dev.	51	50	58	28	78	29	69	47	71

P4	P5	P6	P7	P8	P9
227	186	306	155	61	283
13	110	63	113	84	85
34	53	51	54	61	125
61	134	99	43	58	126
289	142	122	190	383	219
150	202	185	155	118	144
30	61	116	69	43	147
197	64	158	85	87	126
169	56	23	99	73	83
49	66	103	71	51	73
13	53	23	43	43	73
289	202	306	190	383	283
2223	381	1330	442	891	388
122	107	123	103	102	141
97	56	81	49	101	65

- **Same exact software, counted the same way, accepted by the same customer**
- **Largest instance variance is 22:1**
- **Smallest instance variance is 1.5:1**
- **Average variance is ~6:1**
- **All of these variances would be intolerable for cost or schedule**

Bigger (code) is not better – it's bigger (size does matter!)

- **Assuming a constant defect injection rate, bigger code means more defects (some would argue that bigger code increases the defect injection rate), some which are eliminated in reviews if they are conducted, only a few through testing [1] though downstream and more costly; still others escape into the product. Corrections to code tend to beget still more defects.**
- **Bigger code means more code to change when change is introduced, and additional opportunity to inject still more defects. Humphreys has found that small code changes are 40 times more likely to introduce new defects than original development work. [2]**
- **Bigger code is likely the result of less sophisticated design, a sign of other potential issues. Jones notes that delivery defects that originate in requirements and design far outnumber those from coding. [3]**
- **Bigger code is less likely to be a candidate for reuse which introduces another whole series of issues related to product quality and productivity.**
- **Bigger code has implications for software that is heavily constrained by size limits and execution speed.**
- **If you are the customer paying for size of product or developer's time, you might take exception to 10 of 37 largest providers developing 11,493 lines of the code when the ten shortest programmer totals for the same product was merely 5870 lines of code. If you are the customer, it is highly unlikely that your provider will either have this type of data for comparison and even more unlikely that they would share it they knew to look for it!**

1. *Defect Management through the Personal Software Process(SM);* CrossTalk, September 2003
2. *A Discipline for Software Engineering*; Watts Humphrey; Addison-Wesley; 1995 pg. 84
3. *Software Quality: Analysis and Guidelines for Success*; Capers Jones; International Thomson Computer Press; 1997

So, how do we do this better with FPs from planning to deployment?

Get closer to the right size of the house . . .

Elicitation with the customer is a discussion you will have anyhow (the spreadsheet is merely a way to record it)

Do you have organizational measures on which to predict cost and hours / schedule once you have a size?

Do you have multiple ways of estimating that might show you the overlapping space and raise confidence?

Do you contribute your measures to an organizational repository for your benefit and that of others?

Defect re-work is already in your organizational productivity data; what happens if you eliminate much of that re-work?

An example (build a house) with five variance scenarios

- Scenario 1 – Actual cost exceeds estimated cost by $30,000 or 10 percent
- Scenario 2 – Actual size is less than estimated by 150 sq. feet or 5 percent
- Scenario 3 – Actual delivery is 40 days late or 33 percent
- Scenario 4 – Actual size is more than estimated by 150 sq. feet or 10 percent, AND actual deliver is 40 days late or 33 percent
- Scenario 5 – Actual size is more than estimated by 150 feet or 10 percent

Scenario	Estimated Cost	Actual Cost	Estimated Size (sq. ft.)	Actual Size (sq. ft.)	Estimated Delivery (days)	Actual Delivery (days)
1	300,000	330,000	3000	3000		
2	300,000	300,000	3000	2850		
3	300,000	300,000			120	160
4	300,000	300,000	3000	3150	120	160
5	300,000	300,000	3000	3150		

Five scenarios / five outcomes

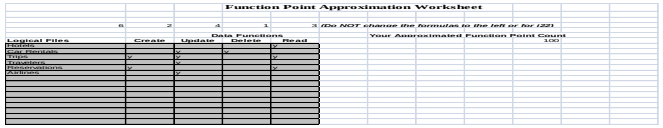
As the customer of this activity, how do you react to the outcomes summarized below?

Scenario	Cost Variance	Size Variance	Schedule Variance	You win or lose?
1	30000			Lose
2		(150)		Lose
3			40	Lose
4		150	40	Uncertain
5		150		Likely win

As the service provider, how do we address these variances throughout the product lifecycle?

Three mitigations are proposed to mitigate these variations through improved estimating and monitoring

1. **Approximate the size as soon as requirements are discovered (not merely when baselined, accepted, or approved). We now have an idea of the size of the product / house.**

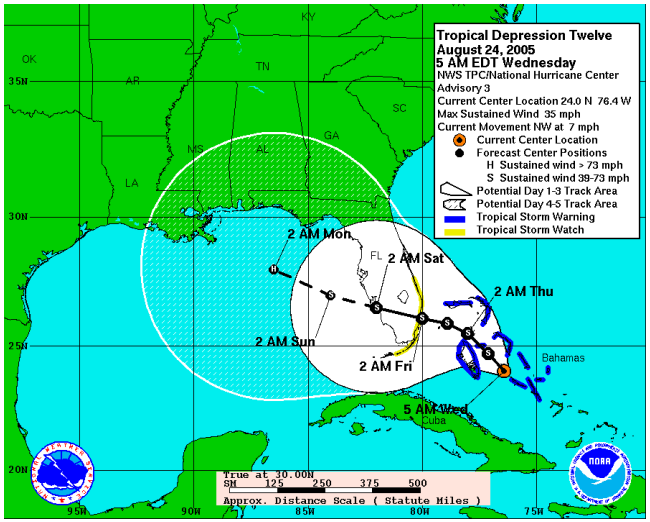

Function Point Approximation Worksheet

Logical Files	Create	Update	Delete	Read
Facility				
Car Rentals				
Trips				
Travelers				
Destinations				
Airlines				

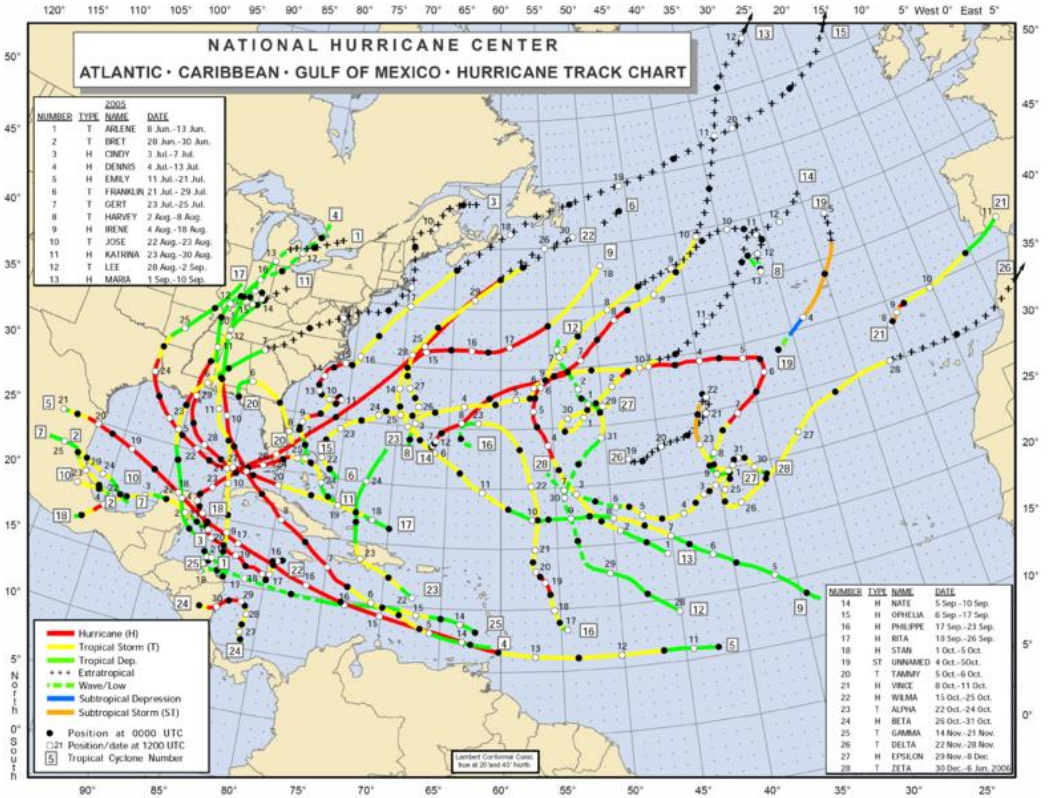
More approximations are preferred – statistical *cone of uncertainty*.

Hurricane Katrina

Provides a likely path (outcome) and margin of error



Use multiple models (QDE) and historic data



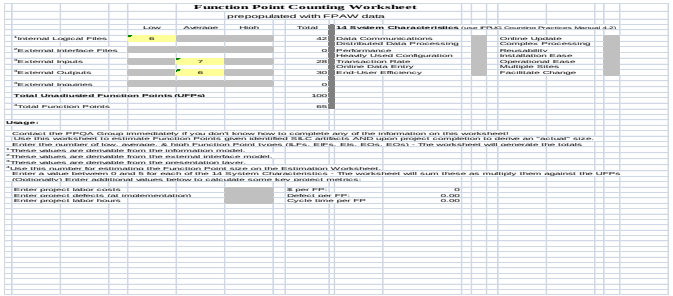
Approximate based on historical performance data (this assumes that such data is collected, stored, and analyzed).

See also CMMI-DEV® v1.2:

- **Measurement and Analysis, SG2**
- **Organizational Process Performance, SG1**

Three mitigations (continued)

2. **Estimate when size is understood and resources are made available to the project**



Function Point Counting Worksheet

prepopulated with FPAW data

	Low	Average	High	Total
*Internal Logical Files	6			42
*External Interface Files				0
*External Inputs		7		28
*External Outputs		6		30
*External Inquiries				0
Total Unadjusted Function Points (UFPs)				100
*Total Function Points				65

14 System Characteristics (use FPAS Counting Practices Manual 4.2)

Data Communications		Online Update	
Distributed Data Processing		Complex Processing	
Performance		Reusability	
Heavily Used Configuration		Installation Ease	
Transaction Rate		Operational Ease	
Online Data Entry		Multiple Sites	
End-User Efficiency		Facilitate Change	

Usage:

Contact the PPQA Group immediately if you don't know how to complete any of the information on this worksheet!
Enter the number of Internal Logical Files, External Interface Files, External Inputs, External Outputs, and External Inquiries that are identified SLC artifacts AND upon project completion to derive an "actual" size.
Enter the number of low, average, & high Function Point Items (ILFs, EIFs, EIs, EOs, EQs) - The worksheet will generate the totals.
*These values are derivable from the information model.
**These values are derivable from the external interface model.
***These values are derivable from the presentation layer.
*Use this number for estimating the Function Point size on the Estimation Worksheet.
Enter a value between 0 and 5 for each of the 14 System Characteristics - The worksheet will sum these as multiply them against the UFPs
(Optionally) Enter additional values below to calculate some key project metrics.

Enter project labor costs		$ per FP:	0
Enter project defects (at implementation)		Defect per FP:	0.00
Enter project labor hours		Cycle time per FP	0.00

3. **Count, record, and analyze the size, cost, and schedule of the project.** **(can be used for future estimations)**

Last question: *How many defects remain in the product?*
(Latent defect estimation)

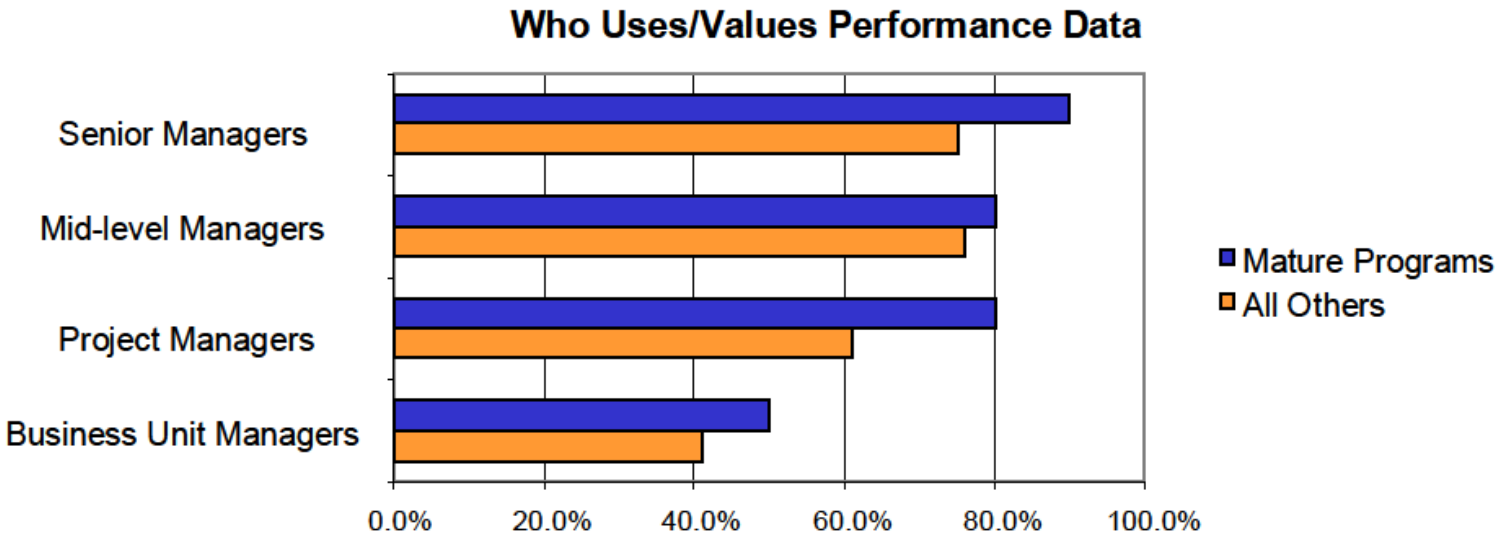
Place a check mark in the intersecting cells for each defect found by each participant.
Count the defects that each engineer found (*Counts* for Engineer A, B, and C).
Column A: check and count all the defects found by the engineer who found the most unique defects. 5
Column B: check and count all of the defects found by all of the other engineers. 4
Column C: check and count the defects common to columns A and B. 2
The estimated number of defects in the product is AB/C. Round to the nearest integer. (5 * 4) / 2 = 10
The number of defects found in the inspection is A+B-C. 5 + 4 – 2 = 7
The estimated number of defects remaining is the estimated number of defects in the product minus the number found. (AB/C) – (A+B-C). 10 – 7 = 3

Use team "thresholds" to determine whether or not to repeat the Peer Review.

Defect No	Engineer Larry	Engineer Curly	Engineer Moe	"Column A"	"Column B"	"Column C"
1	√			√		
2	√			√		
3			√		√	
4	√	√		√	√	√
5	√			√		
6	√		√	√	√	√
7		√			√	
Counts	5	2	2	5	4	2

The capture-recapture method (CRM) has been used for decades by population biologists to accurately determine the number of organisms studied. LaPorte RE, McCarty DJ, Tull ES, Tajima N., Counting birds, bees, and NCDs. Lancet, 1992, 339, 494-5.
See also Introduction to the Team Software Process; Humphrey; 2000; pgs. 345 – 350

Why you should care . . .



2010 DCG Survey Results Performance Measurement; **David Consulting Group; 2010**

Closing thoughts . . .

Get closer to the right size of the product (house) . . .

Elicitation with the customer is a discussion you will have anyhow (the spreadsheet is merely a way to record it)

Do you have organizational measures on which to predict cost and hours / schedule once you have a size?

Do you have multiple ways of estimating that might show you the overlapping space and raise confidence?

Do you contribute your measures to an organizational repository for your benefit and that of others?

What is your latent defect rate? Your requirements volatility rate
(function point analysis can help with both!)

Defect re-work is already in your organizational productivity data; what happens if you eliminate much of that re-work?

Additional References

[1] *Beyond Defect Removal: Latent Defect Estimation with Capture Recapture Method*; CrossTalk; August, 2007

[2] Capers Jones has reported on a survey of organizations that used lines of code as a size for software; one-third of the participants counted comments as lines of code, one-third did not include lines of code in their counts, and the other one-third didn't know if they counted comments or not

[3] *The Statistically Unreliable Nature of Lines of Code;* CrossTalk, April, 2005

[4] ISO / IEC 20926:2009

[5] http://www.ifpug.org/certification/cfps.htm

[6] *Chaos Summary 2009*; Standish Group, 2009

[7] *A Discipline for Software Engineering*; Watts Humphrey; Addison-Wesley; 1995 pg. 84

[8] *Why Software Fails*; Robert N. Charette; IEEE Spectrum; September, 2005

[9] *Counting Lines of Code: Virtually Worthless for Estimating and Software Sizing*, IT Metrics and Productivity Journal; December, 2009

[10] *Is There a Weakest Link After All?,* IT Metrics and Productivity Journal; December, 2009

[11] *Is There Value to using Lines of Code for Measuring People After All?,* IT Metrics and Productivity Journal; December, 2009

[12] *2010 DCG Survey Results Performance Measurement;* David Consulting Group; 2010