Using Function Point Metrics For Software Economic Studies

Capers Jones, Chief Scientist Emeritus

Email: CJonesiii@cs.com
Web://www.spr.com

June 27, 2008

REASONS FOR SUCCESS OF FUNCTION POINTS

- **Function points match standard economic definitions for productivity analysis:**

  "Goods or services produced per unit of labor or expense."

- **Function points do not distort quality and productivity as do "Lines of Code" or LOC metrics.**

- **Function points support activity-based cost analysis, baselines, benchmarks, quality, cost, and value studies.**

REASONS FOR SUCCESS OF FUNCTION POINTS

- **Lines of code metrics penalize high-level programming languages.**

- **If used for economic studies with more than one language LOC metrics should be considered *professional malpractice*.**

- **Cost per defect metrics penalize quality and make buggy software look best. For quality economic studies cost per defect metrics are invalid. Function points are best.**

- **Function point metrics have the widest range of use of any software metric in history: they work for both economic and quality analyses.**

MAJOR FUNCTION POINT USES CIRCA 2008

- **Function points are now a standard sizing metric.**
- **Function points are now a standard productivity metric.**
- **Function points are now a powerful quality metric.**
- **Function points are now a powerful schedule metric.**
- **Function points are now a powerful staffing metric.**
- **Function points are now used in software litigation.**
- **Function points are now used for outsource contracts.**
- **Function points can be used for cost analysis (with care).**
- **Function points can be used for value analysis (with care.)**

NEW FUNCTION POINT USES CIRCA 2009

- **Function points used for portfolio analysis.**
- **Function points used for backlog analysis.**
- **Function points used for risk analysis.**
- **Function points used for real-time requirements changes.**
- **Function points used for software usage studies.**
- **Function points used for delivery analysis.**
- **Function points used for COTS analysis.**
- **Function points used for occupation group analysis.**
- **Function points used for maintenance and ITIL analysis.**

FUNCTION POINTS FROM 2009 TO 2019

- **Resolve functional vs. technical requirements issues.**
- **Resolve overhead, inflation, and cost issues.**
- **Resolve global variations in work hours and work days.**
- **Resolve issue of > 90 software occupation groups**
- **Produce conversion rules for function point variations.**
- **Improve cost, speed, and timing of initial sizing.**
- **Develop and certify “micro function points.”**
- **Expand benchmarks to > 25,000 projects.**

INDUSTRY EVOLUTION CIRCA 2008-2018

- **Moving from software development to software delivery**
- **Development rates < 25 function points per staff month**
- **Delivery rates > 500 function points per staff month**
- **Delivery issues: reuse taxonomy, quality, security, band width**
- **Delivery methods:**

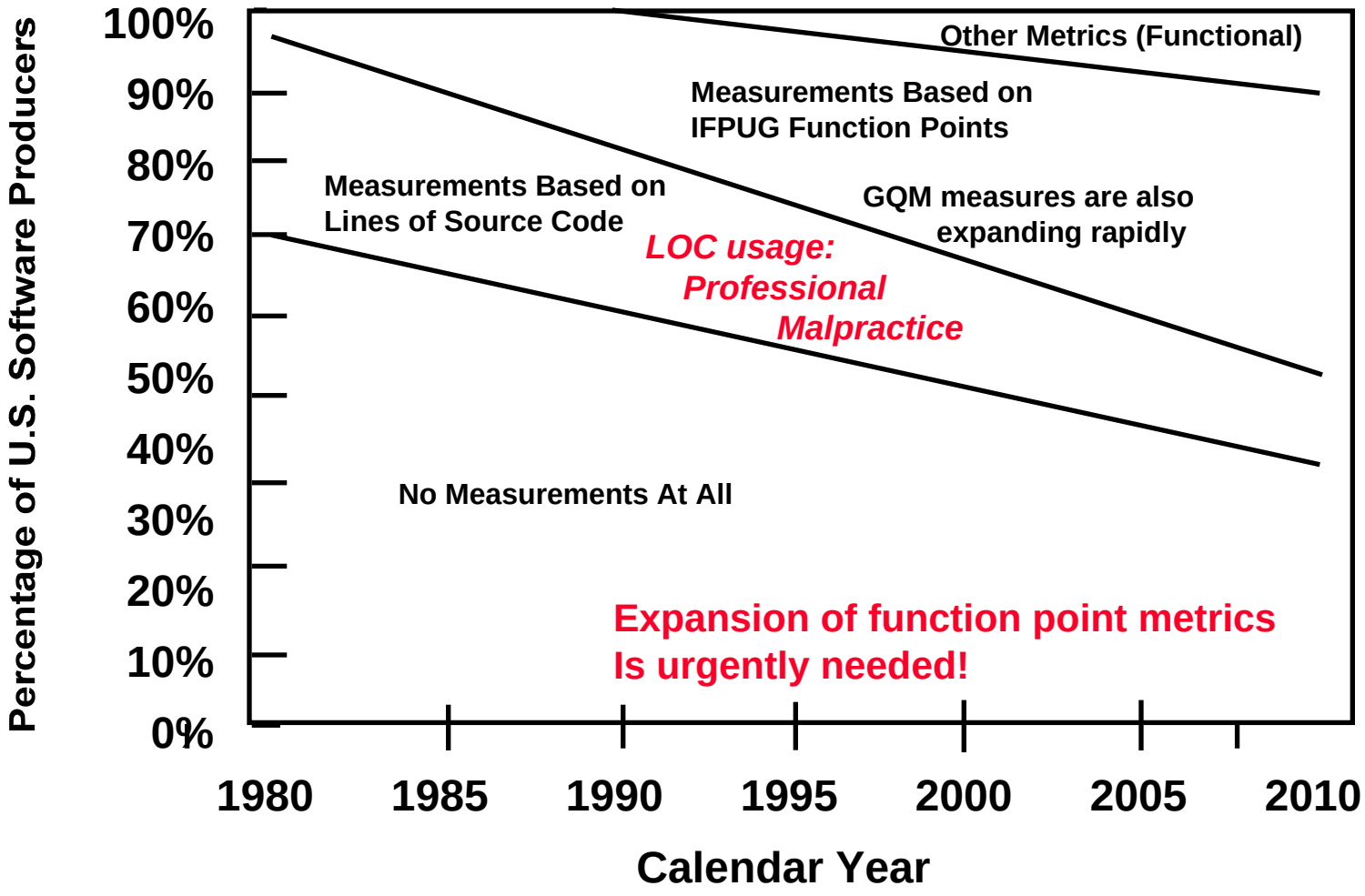
  Service Oriented Architecture (SOA)
  Software as a Service (SaaS)
  Commercial reusable libraries
  Cloud computing

FUNCTION POINT EVOLUTION CIRCA 2008-2018

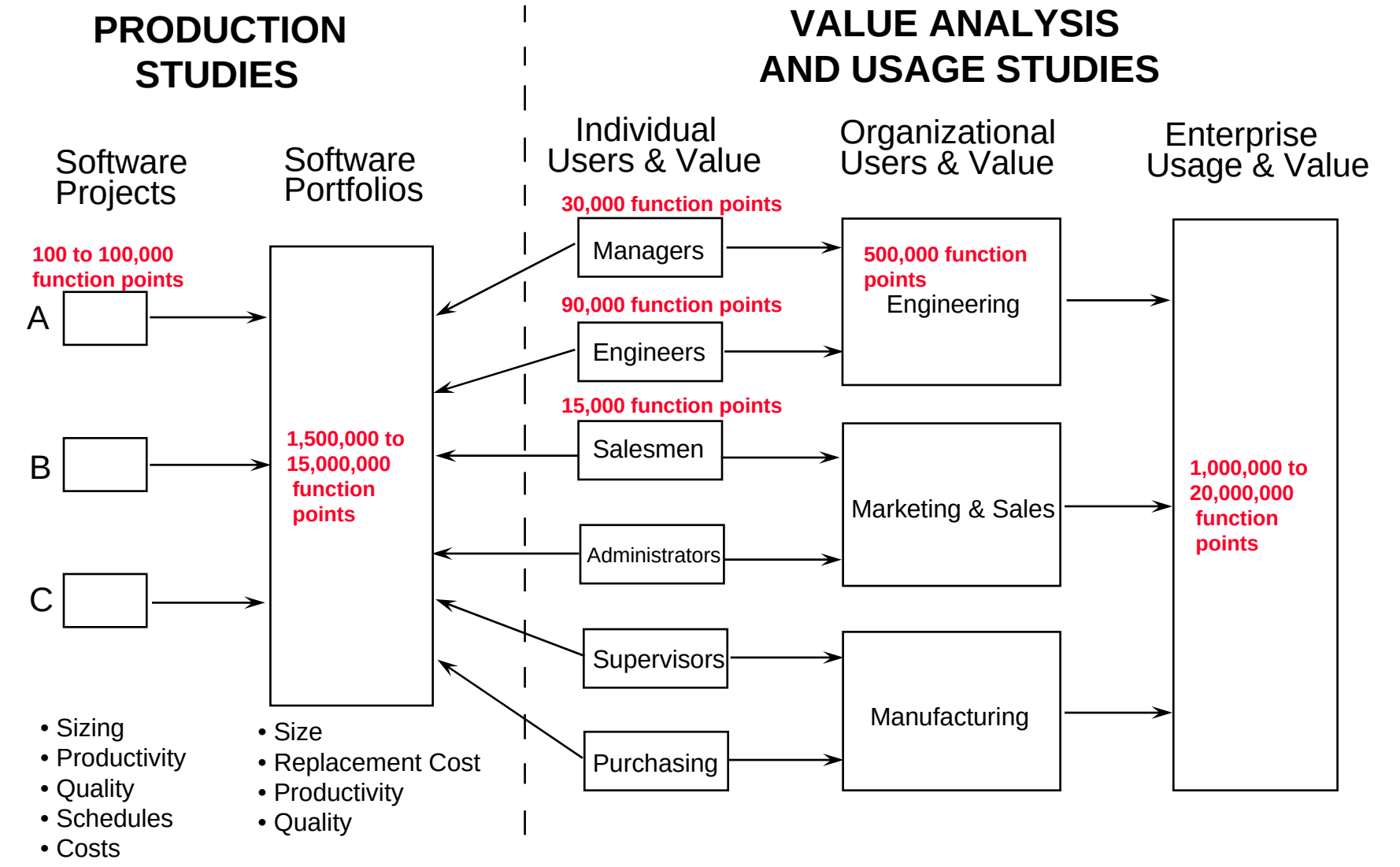
- **Measure delivered features as well as development.**
- **Measure and catalog reusable features using formal taxonomy.**
- **Measure deployment, installation, and usage.**
- **Measure quality and security.**
- **New measurements needed:**

  Sizes of reusable components and COTS packages
  Quality, security of reusable components
  Sizes of delivered applications
  Deployment and usage of delivered applications

MIGRATION TO FUNCTIONAL METRICS



FUNCTIONAL METRICS IN INDUSTRY



ESTIMATED U.S. SOFTWARE CIRCA 2008

Size of Application	Number of Applications
1	630,000
10	320,000
100	131,500
1,000	70,500
10,000	21,750
100,000	185
1,000,000	30
TOTAL	1,173,965

- **Small updates are most numerous.**
- **Large applications are most hazardous and expensive.**

ESTIMATED FUNCTION POINT PENETRATION

Size of Application	Function Point Usage
1	0.00%
10	0.00%
100	25.00%
1,000	15.00%
10,000	5.00%
100,000	0.00%
1,000,000	0.00%

- **Function point rules do not work < 15 function points.**
- **Counting is too slow and costly > 15,000 function points.**

ESTIMATED PROJECTS COUNTED CIRCA 2008

Size of Application	Function Point Usage
1	0
10	0
100	32,875
1,000	10,575
10,000	652
100,000	0
1,000,000	0
TOTAL	44,102

- **Function point usage declines with application size.**
- **Function points are not used < 15 function point size limit.**

FUNCTION POINT EXPANSION CIRCA 2009

Function points need to extend their range of use:

- **"Micro function points" are needed for small updates below 15 function points in size.**
- **About 20% of all software effort goes to small updates.**
- **Function point costs and speed need improvement to work effectively > 15,000 function points in size.**
- **More than 45% of all effort goes to large applications.**
- **Function point range should run from 1 to 1,000,000 function points.**

ONE YEAR FOR 10,000 FUNCTION POINTS

Starting size = 10,000 function points

Task	Size (FP)	Staff	Effort	Cost
Large	500	3.33	33.33	$250,000
Small	90	1.00	5.29	$39,706
V. Small	180*	1.20	10.59	$79,412
Bugs	1750*	7.29	87.50	$656,750
TOTAL	2520	15.00	136.72	$1,025,368

* Can't be measured using standard function points

THE NEED FOR "MICRO FUNCTION POINTS"

About 70% of small updates can't be measured

Small changes run from 0.001 to 15 FP

Each change is small, but total > 25,000 per year

Total volume per year > 100,000 function points

Micro function points are needed to expand usage:

- **New counting rules with negative adjustments**
- **Backfiring**
- **Pattern matching**

FUNCTION POINT COUNTING CIRCA 2008

Speed of count = 400 to 600 FP per day

Consulting cost = $3,000 per day

Cost range = $5.00 to $7.50 per function point

- **High costs and low speed limit usefulness of function points to small or mid-sized software applications**

FUNCTION POINT EXPANSION CIRCA 2009

Daily Counting Speed	Percent of Projects Counted	Maximum Size Counted
500	20%	15,000
1,000	30%	25,000
5,000	45%	50,000
10,000	75%	75,000
50,000	100%	350,000

MAJOR APPLICATION SIZES CIRCA 2009

Applications	Approximate Size in Function Points
Star Wars Missile Defense	350,000
ERP (SAP, Oracle, etc.)	300,000
Microsoft Vista	159,000
Microsoft Office	98,000
Airline Reservations	50,000
NASA space shuttle	25,000

- Major applications are too large for low-speed counting

ESTIMATED SOFTWARE FAILURE RATES

Size of Application	% Cancelled or delayed
1	0.00%
10	0.01%
100	1.00%
1,000	10.00%
10,000	35.00%
100,000	75.00%
1,000,000	95.00%

- **Major applications fail more often than they succeed!**
- **Risk of failure correlates with increasing application size!**
- **Major applications need fast, early function point analysis!**

RESULTS FOR 10,000 FUNCTION POINTS

No formal risk analysis =	**35% chance of failure**
Risk analysis after requirements =	**15% chance of failure**
Risk analysis before requirements =	**5% chance of failure**

- **Early sizing and formal risk analysis raise the chances of success for large software projects.**

FUNCTION POINTS AND REQUIREMENTS

Requirements change at 1% to 3% per month during development.

Requirements change at 5% to 8% per year after deployment.

Some large systems are used for more than 25 years.

Size at end of requirements	**= 10,000**	**Function points**
Size at first delivery	**= 13,000**	**“ “**
Size after 5 years of usage	**= 18,000**	**“ “**
Size after 25 years of usage	**= 25,000**	**“ “**

Sizing should be continuous from requirements to retirement.

Continuous sizing needs low cost, high speed function points.

FUNCTION POINTS AND ECONOMIC STUDIES

Function points are the only metric that can study all Software activities without exception.

There are two key dimensions when applying function points to economic analysis:

Assignment Scope

"The amount of work assigned to a specific employee."

Production Rate

"The amount of work completed in a specific time period."

ASSIGNMENT AND PRODUCTION EXAMPLES

Application = 1,000 Function points

Activity	Assignment Scope	Production Rate (Monthly)
Requirements	500	200
Design	500	175
Coding	200	25
Testing	150	40
Documentation	1000	150
Management	1000	40
OVERALL	222	9.25

ASSIGNMENT AND PRODUCTION EXAMPLES

Application = 1,000 Function points

Activity	Staff	Effort (months)	Schedule (months)
Requirements	2	5	2.5
Design	2	6	3.0
Coding	5	40	8.0
Testing	7	25	7.0
Documentation	1	7	7.0
Management	1	25	24.0
OVERALL	4.5	108	24.0

USEFUL FUNCTION POINT RULES OF THUMB

- **Function points ^ 0.40 power = calendar months in schedule**
- **Function points ^ 1.15 power = pages of paper documents**
- **Function points ^ 1.20 power = number of test cases**
- **Function points ^ 1.25 power = software defect potential**
- **Function points / 150 = development technical staff**
- **Function points / 1,500 = maintenance technical staff**

NOTE: These rules assume IFPUG Version 4.2 counting rules.

FUNCTION POINTS AND USAGE ANALYSIS

Occupation	Function Points Available	Software Packages	Daily usage (hours)
1 Military planners	5,000,000	30	7.5
2 Physicians	3,000,000	20	3.0
3 FBI Agents	1,500,000	15	3.5
4 Attorneys	325,000	10	4.0
5 Air-traffic controllers	315,000	3	8.5
6 Accountants	175,000	10	5.0
7 Pharmacists	150,000	6	4.0
8 Electrical engineers	100,000	20	5.5
9 Software engineers	50,000	20	7.0
10 Project managers	35,000	10	1.5

Computers and software are now vital tools for all knowledge-based occupations. Usage contributes to software value.

PERSONAL FUNCTION POINTS

Products	Circa 2008 Available	Circa 2018 Available	Daily usage (hours)
1 Home computer	1,000,000	2,000.000	2.5
2 Automobile	300,000	750,000	3.0
3 Smart appliances	100,000	750.000	24.0
4 Televisions	25,000	125,000	4.0
5 Home alarms	5,000	15,000	24.0
6 Home music	7,500	20,000	2.5
7 I-Phone	20,000	30.000	3.0
8 Digital camera	2,000	5,000	0.5
9 Electronic books	10,000	20,000	2.5
10 Social networks	25,000	75,000	2.5
TOTAL	1,494,500	3,790,000	20.5

Personal software is expanding rapidly in all developed countries.

FUNCTION POINTS AND MANAGEMENT TOOLS

SELECTED TOOLS	Lagging	Average	Leading
1 Project planning	1,000	1,500	3,000
2 Cost estimating	--	300	3,000
3 Project office	--	--	3,000
4 Statistical analysis	--	750	3,000
5 Personnel mgt.	500	1,000	2,000
6 Quality estimating	--	--	2,000
7 Process Assessment	--	500	2,000
8 Risk analysis	--	--	1,500
9 Value analysis	--	250	1,500
10 Department budgets	500	700	1,000
TOTALS	2,000	5,000	22,000
Tools used	3	7	10

FUNCTION POINTS AND SOFTWARE TOOLS

SELECTED AREAS	Lagging	Average	Leading
1 Project mgt.	2,000	5,000	22,000
2 Quality Assur.	0	2,500	10,000
3 Development	14,000	19,000	25,000
4 Maintenance	2,500	10,500	20,000
5 Testing	2,000	4,000	15,000
6 Documentation	3,000	5,000	12,000
7 Training	2,000	4,500	10,000
8 Nationalization	0	3,000	7,500
9 Customer supp.	2,500	5,500	10,500
10 Change control	3,000	6,000	12,000
TOTALS	31,000	65,000	144,000
Tools used	20	35	75

PORTFOLIO FUNCTION POINT ANALYSIS

Size in terms of IFPUG function points

Unit	In-house	COTS	Open-Source	TOTAL
Headquarters	500,000	500,000	50,000	1,050,000
Sales/Market.	750,000	750,000	75,000	1,575,000
Manufacture.	1,500,000	500,000	100,000	2,100,000
Acct/Finance	500,000	1,000,000	10,000	1,510,000
Research	750,000	250,000	150,000	1,150,000
Cust. Support	300,000	250,000	50,000	600,000
Human Res.	100,000	250,000		350,000
Purchasing	250,000	500,000		750,000
Legal	200,000	750,000		950,000
TOTAL	4,850,000	4,750,000	435,000	10,035,000

PORTFOLIO ANALYSIS WITH FUNCTION POINTS

Software portfolios are major business assets.

Software portfolios are large and growing: > 10,000,000 FP

Portfolios include:

- **IT, systems, embedded, and web applications**
- **In-house applications**
- **Outsourced applications**
- **Commercial applications (COTS)**
- **Open-source applications**
- **SOA and SaaS**

NEW USES FOR FUNCTION POINTS

Usage analysis is a key factor in value analysis.

Portfolio analysis is a key factor in market planning.

Portfolio analysis is a key factor in both corporate and government operations.

When function points are used for development, delivery, maintenance, usage studies, and portfolio analysis they will become a major metric for global economic studies.

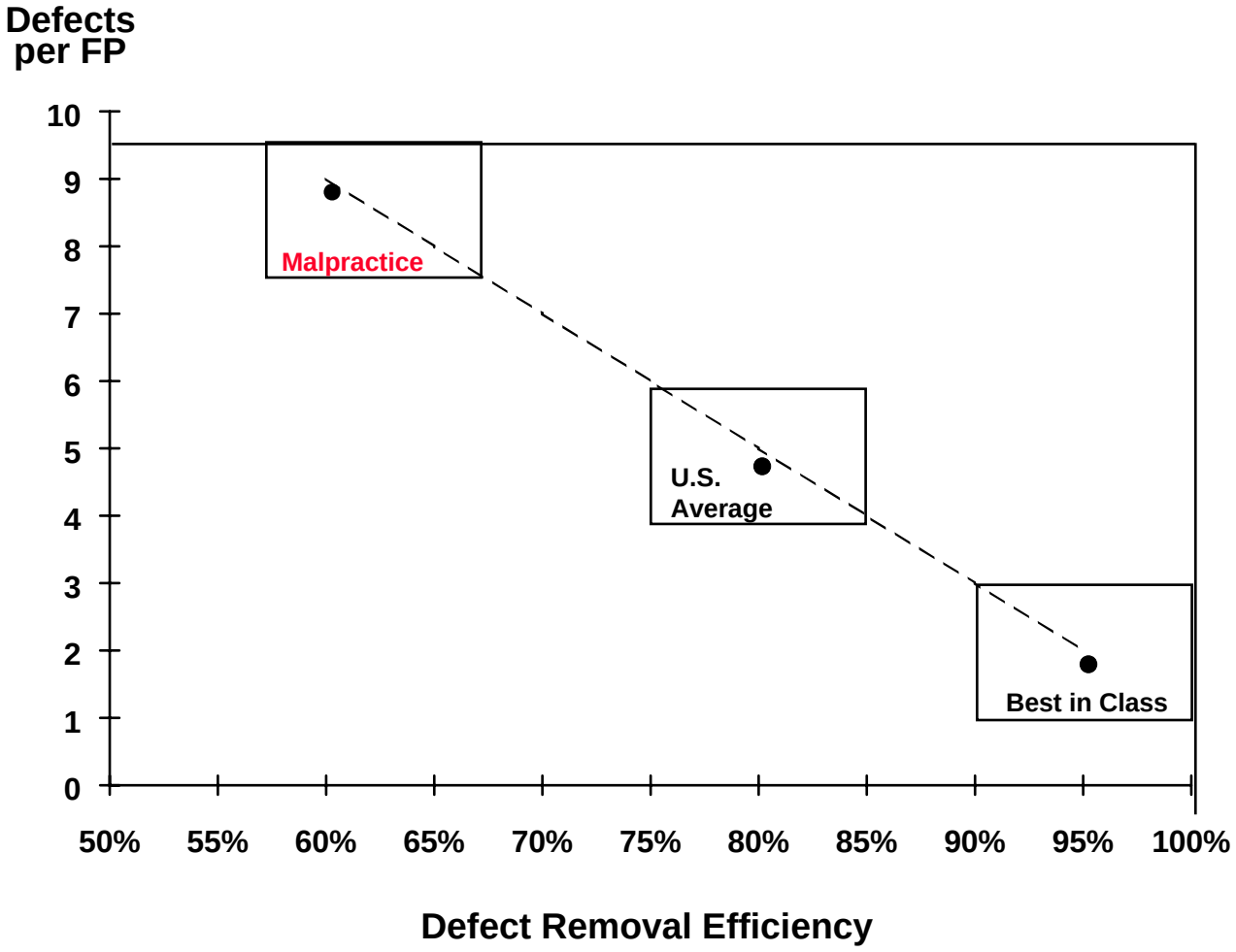
Usage analysis and portfolio analysis deal with millions of Function points. Speed and cost of counting are barriers that must be overcome.

U. S. SOFTWARE QUALITY AVERAGES

(Defects per Function Point))

	System Software	Commercial Software	Information Software	Military Software	Overall Average
Defect Potentials	6.0	5.0	4.5	7.0	5.6
Defect Removal Efficiency	94%	90%	73%	96%	88%
Delivered Defects	0.4	0.5	1.2	0.3	0.65
First Year Discovery Rate	65%	70%	30%	75%	60%
First Year Reported Defects	0.26	0.35	0.36	0.23	0.30

SOFTWARE QUALITY RANGES



U.S. AVERAGES FOR SOFTWARE QUALITY

(Data expressed in terms of defects per function point)

Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	1.00	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Documents	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
TOTAL	5.00	85%	0.75

(Function points show all defect sources - not just coding defects)

BEST IN CLASS SOFTWARE QUALITY

(Data expressed in terms of defects per function point)

Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	0.40	85%	0.08
Design	0.60	97%	0.02
Coding	1.00	99%	0.01
Documents	0.40	98%	0.01
Bad Fixes	0.10	95%	0.01
TOTAL	2.50	96%	0.13

OBSERVATIONS

- **Most often found above Level 3 on the SEI CMM scale**
- **Most often found in systems and military software**

THIRTEEN SOFTWARE METRICS CRITERIA

A Useful Software Metric Should:

1. Be standardized
2. Be unambiguous
3. Have a formal user group
4. Have adequate published data
5. Have tools available for new projects
6. Have tools available for legacy projects
7. Have conversion rules for related metrics
8. Deal with all deliverables
9. Support all kinds of software
10. Support all programming languages plus mixed
11. Support all sizes of applications
12. Support new + reused artifacts
13. Be cost-effective and fast to apply

CRITERIA FOR SOFTWARE METRICS SELECTION

IFPUG Function Point Metric Meets 11 of 13 Criteria:

1. Are standardized
2. Are unambiguous
3. Has a formal user group
4. Has adequate published data
5. Has tools available for new projects
6. Has tools available for legacy projects
7. Has conversion rules for related metrics
8. Deals with all deliverables
9. Supports all kinds of software
10. Supports all programming languages plus mixed languages
11. Counting inaccurate < 15 function points
12. Can be used with reused artifacts
13. Are somewhat costly and slow to apply

CRITERIA FOR SOFTWARE METRICS SELECTION

Pattern Matching Meets 9 of 13 Criteria:

1. Are not standardized
2. Are unambiguous (**with exceptions**)
3. Has no formal user group
4. Has little published data
5. Has tools available for new projects
6. Has tools available for legacy projects
7. Has conversion rules for related metrics
8. Deals with all deliverables
9. Supports all kinds of software
10. Supports all programming languages plus mixed languages
11. Counting accurate < 15 function points
12. Can be used with reused artifacts
13. Is very fast and inexpensive, but of uncertain accuracy

CRITERIA FOR SOFTWARE METRICS SELECTION

Legacy Function Point Metrics Meets 8 of 13 Criteria:

1. Are not standardized
2. Are unambiguous
3. Has no formal user group
4. Has no published data
5. Has no tools available for new projects
6. Has tools available for legacy projects
7. Has conversion rules for related metrics
8. Deals with all deliverables
9. Supports primarily IT projects; not embedded software
10. Supports some legacy programming languages
11. Counting inaccurate < 15 function points
12. Can be used with reused artifacts
13. Is very fast and inexpensive

CRITERIA FOR SOFTWARE METRICS SELECTION

Function Point Variants (COSMIC etc.) meet 7 criteria:

1. Are standardized (conflicting standards exist)
2. Are unambiguous (with exceptions)
3. Have formal user groups
4. Lack adequate published data
5. Have tools available for new projects
6. Have tools available for legacy projects
7. Lack conversion rules for related metrics
8. Deal with all deliverables
9. Support all kinds of software
10. Support all programming languages plus mixed languages
11. Counting inaccurate < 15 function points
12. Can be used with reused artifacts
13. Are not particularly fast or inexpensive

CRITERIA FOR SOFTWARE METRICS SELECTION

The Backfiring Method Meets 7 of 13 Criteria:

1. Is **not** standardized
2. Is **very** ambiguous
3. Has **no** formal user groups
4. Has **inconsistent** published data
5. Has no tools available until code is complete
6. Has tools available for legacy projects
7. Has conversion rules for related metrics
8. Deals with all deliverables
9. Supports all kinds of software
10. Supports **procedural** programming languages; plus mixed
11. Can be used < 15 function point size limit
12. Can be used for reused artifacts
13. Is very fast and inexpensive

CRITERIA FOR SOFTWARE METRICS SELECTION

The Logical Statement Metric Meets 6 of 13 Criteria:

1. Is **not** standardized (conflicts between IEEE, SPR, and SEI)
2. Is **somewhat** ambiguous and may be incorrect
3. **Does not** have a formal user group
4. **Does not** have adequate published data (data is misleading)
5. Has tools available for new projects
6. Has tools available for legacy projects
7. Has conversion rules for related metrics
8. **Does not** deal with all deliverables
9. Does support all kinds of software
10. Supports **procedural** programming languages; mixed is possible
11. Does support all application sizes
12. Can be used with reusable code artifacts
13. Is fast, but difficult to count accurately without automated tools

CRITERIA FOR SOFTWARE METRICS SELECTION

The Physical LOC Metric Meets 5 of 13 Criteria:

1. Is **not** standardized
2. Is **very** ambiguous with multiple languages and is often incorrect
3. **Does not** have a formal user group
4. **Does not** have adequate published data (data is misleading)
5. Has tools available for new projects
6. Has tools available for legacy projects
7. **Does not** have conversion rules for related metrics
8. **Does not** deal with all deliverables
9. Does support all kinds of software
10. **Does not** support all or mixed programming languages (no VB)
11. Does support all sizes of software
12. **Hard to use** with reusable code artifacts
13. Is fast but difficult to count accurately without automated tools

CRITERIA FOR SOFTWARE METRICS SELECTION

Goal-Question Metrics Meet 3 of 13 Criteria:

1. Is **not** standardized
2. Is often unambiguous
3. **Does not** have a formal user group
4. Does have adequate published data
5. Does **not** have tools available for new projects
6. **Does not** have tools available for legacy projects
7. **Does not** have conversion rules for related metrics
8. Can be designed for use with all deliverables
9. **Does not** support all kinds of software
10. **Does not** support all or mixed programming languages
11. Does support all application sizes
12. **Partial support** for reusable artifacts
13. Is somewhat slow and expensive to use

CRITERIA FOR SOFTWARE METRICS SELECTION

Object-Oriented Metrics Meet 2 of 13 Criteria:

1. Is **not** standardized
2. Is **somewhat** unambiguous
3. **Does not** have a formal user group
4. **Does not** have adequate published data
5. Does have tools available for new projects
6. **Does not** have tools available for legacy projects
7. **Does not** have conversion rules for related metrics
8. **Cannot** deal with all deliverables
9. **Does not** support all kinds of software (**ONLY** supports OO)
10. **Does not** support all or mixed programming languages
11. Does support all application sizes
12. **Partial support** for reusable artifacts
13. Is somewhat slow and expensive to use

CRITERIA FOR SOFTWARE METRICS SELECTION

Agile Metrics (Story points, RTF) Meet 1 of 13 Criteria:

1. Is **not** standardized
2. Is **somewhat** unambiguous
3. **Does not** have a formal user group
4. **Does not** have adequate published data
5. Does have tools available for new projects
6. **Does not** have tools available for legacy projects
7. **Does not** have conversion rules for related metrics
8. **Cannot** deal with all deliverables
9. **Does not** support all kinds of software (only supports Agile)
10. **Does not** support all or mixed programming languages
11. **Does not** support all application sizes (< 5000 function points)
12. **Partial support** for reusable artifacts
13. Is somewhat slow and expensive to use

CRITERIA FOR SOFTWARE METRICS SELECTION

Cost per Defect metrics Meet 0 of 13 Criteria:

1. Is **not** standardized
2. Is **highly** unambiguous
3. **Does not** have a formal user group
4. **Does not** have adequate published data
5. **Does not** have tools available for new projects
6. **Does not** have tools available for legacy projects
7. **Does not** have conversion rules for related metrics
8. **Cannot** deal with all deliverables
9. **Does not** support all kinds of software (cannot handle zero-defects)
10. **Does not** support all or mixed programming languages
11. **Does not** support all application sizes or deliverables
12. **Partial support** for reusable artifacts
13. Is somewhat slow and expensive to use; **penalizes quality**

METRICS CONVERSION - A CRITICAL GAP

N-DIRECTION METRICS CONVERSION IS NEEDED BETWEEN:

- IFPUG VERSIONS 1, 2, 3, 4, 4.1, and 4.2 COUNTING RULES
- CURRENT IFPUG, MARK II, 3D, COSMIC, AND OTHER VARIATIONS
- FUNCTION POINTS AND PHYSICAL LOC METRICS
- FUNCTION POINTS AND LOGICAL STATEMENT METRICS
- PHYSICAL LOC AND LOGICAL STATEMENTS
- OBJECT-ORIENTED, AGILE, GQM METRICS AND OTHERS

ALL OTHER SCIENTIFIC AND ENGINEERING METRICS HAVE PUBLISHED CONVERSION RULES BETWEEN VARIANT METRICS.

COMPLEXITY - A TECHNICAL GAP

FUNCTION POINT COMPLEXITY IS DECOUPLED FROM:

- COMBINATORIAL COMPLEXITY
- COMPUTATIONAL COMPLEXITY
- CYCLOMATIC COMPLEXITY
- ESSENTIAL COMPLEXITY
- FLOW COMPLEXITY
- SYNTACTIC COMPLEXITY

OF THE 24 KINDS OF COMPLEXITY NOTED IN ENGINEERING AND SCIENTIFIC STUDIES, NONE ARE CURRENTLY INCLUDED IN FUNCTION POINT COUNTS.

STRENGTHS AND WEAKNESSES OF FUNCTION POINTS

The main strengths of IFPUG function point metrics are:

1. Function points stay constant regardless of programming languages used
2. Function points are a good choice for full life-cycle analysis
3. Function points are a good choice for benchmarks and economic studies
4. Function points are supported by many software estimating tools
5. Function points can be converted into logical code statements

The main weaknesses of IFPUG function point metrics are:

1. Accurate counting requires certified function point specialists
2. Function point counting can be time-consuming and expensive
3. There is a need for automated function point counts from requirements
4. There is a need for automated function point counts from code
5. IFPUG has no rules dealing with backfiring
6. IFPUG needs “micro function points” for small updates

STRENGTHS AND WEAKNESSES OF VARIANTS

The main strengths of function point variants are:

1. Variants stay constant regardless of programming languages used
2. Variants are a good choice for full life-cycle analysis

The main weaknesses of function point variants are:

1. Accurate counting requires certified function point specialists
2. Variant counting can be time-consuming and expensive
3. Variants have little published benchmark data
3. Variant counts are erratic for projects below 15 function points
4. Variants lack rules for backfiring
5. Variants lack conversion rules to IFPUG function points:
   3D function points, Mark II function points; Object points;
   COSMIC function points; full function points; Story points; etc.

IT IS NOT IFPUG'S JOB TO CREATE CONVERSION RULES!

STRENGTHS AND WEAKNESSES OF BACKFIRING

The main strengths of backfiring are:

1. A fairly convenient method for aging legacy applications
2. One method for small applications < 15 function points
3. Costs less than < $0.25 per function point
4. Backfiring is supported by many software estimating tools
5. Function points can be converted into code statements and vice versa

The main weaknesses of backfiring are:

1. Accuracy of backfiring is never equal to counts by function point specialists
2. Backfiring varies between physical LOC and logical statements
3. Backfiring varies between individual programming styles
4. Backfiring varies between published lists for same languages
5. Backfiring is not endorsed by any function point association
6. Backfiring research is sparse and literature is scarce
7. Backfiring needs adjustment for dead code and complexity

BACKFIRING: RATIOS OF LOGICAL SOURCE CODE STATEMENTS TO FUNCTION POINTS

Language	Nominal Level	Source Statements Per Function Point		
		Low	Mean	High
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
FORTRAN	3.00	75	107	160
COBOL	3.00	65	107	150
PASCAL	3.50	50	91	125
PL/I	4.00	65	80	95
Ada 83	4.50	60	71	80
C++	6.00	30	53	125
Ada 95	6.50	28	49	110
Visual Basic	10.00	20	32	37
SMALLTALK	15.00	15	21	40
SQL	27.00	7	12	15

Note: Assumes Version 4 of IFPUG Counting Rules

STRENGTHS AND WEAKNESSES OF PATTERN MATCHES

The main strengths of pattern matching are:

1. Only convenient method for applications > 100,000 function points
2. Also useful for small applications < 15 function points
3. Fastest, cheapest known approach for sizing software
4. Costs less than $0.01 per function point
4. Pattern matching takes < 1 minute to size any known application
5. Only known method for sizing software with hidden requirements
6. Only known method for sizing COTS or Open Source by customers

The main weaknesses of pattern matching are:

1. Requires large volumes of historical data to get started
2. Requires a full taxonomy of software classes and types
3. Pattern matching is experimental in 2008
4. Pattern matching is proprietary in 2008
5. Pattern matching is not endorsed by any function point association

EXAMPLES OF SIZING WITH PATTERN MATCHING

Application	Size in IFPUG Function Points
Star Wars Missile Defense	372,086
SAP	296,574
Microsoft Vista	159,159
Microsoft Office Professional 2007	97,165
FBI Carnivore	24,242
Skype	21,202
Apple I Phone	19,366
Google Search engine	18,640
Linux	17,505

STRENGTHS AND WEAKNESSES OF LOGICAL STATEMENTS

The main strengths of logical statements are:

1. Logical statements exclude dead code, blanks, and comments
2. Logical statements can be converted into function point metrics
3. Logical statements are used in a number of software estimating tools

The main weaknesses of logical statements are:

1. Logical statements can be difficult to count
2. Logical statements are not extensively automated
3. Logical statements are a poor choice for full life-cycle studies
4. Logical statements are ambiguous for some “visual” languages
5. Logical statements may be ambiguous for software reuse
6. Logical statements maybe erratic for conversion to the physical LOC metric
7. Logical statements are professional malpractice for software economics

STRENGTHS AND WEAKNESSES OF PHYSICAL LINES OF CODE (LOC)

The main strengths of physical lines of code (LOC) are:

1. The physical LOC metric is easy to count
2. The physical LOC metric has been extensively automated for counting
3. The physical LOC metric is used in some software estimating tools

The main weaknesses of physical lines of code are:

1. The physical LOC metric may include substantial “dead code”
2. The physical LOC metric may include blanks and comments
3. The physical LOC metric is ambiguous for mixed-language projects
4. The physical LOC metric is a poor choice for full life-cycle studies
5. The physical LOC metric does not work for some “visual’ languages
6. The physical LOC metric is erratic for backfiring to function points
7. The physical LOC metric is professional malpractice for economics

STRENGTHS AND WEAKNESSES OF OO METRICS

The main strengths of OO metrics are:

1. The OO metrics are psychologically attractive within the OO community
2. The OO metrics can distinguish simple from complex OO projects

The main weaknesses of OO metrics are:

1. The OO metrics do not support studies outside of the OO paradigm
2. The OO metrics do not deal with full life-cycle issues
3. The OO metrics have no conversion rules to lines of code metrics
4. The OO metrics have no conversion rules to function point metrics
5. The OO metrics lack automation for counting
6. The OO metrics are not supported by software estimating tools
7. The OO metrics have no benchmarks or support organization

STRENGTHS AND WEAKNESSES OF AGILE METRICS

The main strengths of Agile metrics are:

1. Agile metrics are useful within the Agile community
2. Agile metrics can create Agile benchmarks but have no yet done so
3. Agile metrics include RTF, Story Points, and LOC metrics

The main weaknesses of Agile metrics are:

1. Agile metrics do not support studies outside of the Agile paradigm
2. Agile metrics do not deal with full life-cycle issues
3. Agile metrics have no conversion rules to lines of code metrics
4. Agile metrics have no conversion rules to function point metrics
5. Agile metrics lack automation for counting
6. Agile metrics are not supported by software estimating tools
7. Agile metrics have no benchmarks available

STRENGTHS AND WEAKNESSES OF GQM METRICS

The main strengths of Goal-Question-Metrics (GQM) are:

1. GQM metrics are useful for a wide range of problems
2. GQM metrics are useful for executive reporting
3. GQM metrics support innovative solutions

The main weaknesses of GQM metrics are:

1. GQM metrics are unique, and difficult to use for benchmarks
2. GQM metrics collect random kinds of data
3. GQM metrics have no conversion rules to lines of code metrics
4. GQM metrics have no conversion rules to function point metrics
5. GQM metrics lack automation for counting
6. GQM metrics are not supported by software estimating tools
7. GQM metrics have no user group or support organization

STRENGTHS AND WEAKNESSES OF COST PER DEFECT

The main strengths of cost per defect metrics are:

1. Cost per defect is easy to count

The main weaknesses of cost per defect metrics are:

1. **Cost per defect metrics penalize quality!**
2. Cost per defect metrics ignore fixed costs
3. Cost per defect metrics violate standard economic principles
4. Cost per defect has no conversion rules to function point metrics
5. Cost per defect lacks automation for counting
6. Cost per defect has no user group or support organization
7. Cost per defect is professional malpractice for quality analysis

FUNCTION POINTS ALONE ARE NOT ENOUGH!!

To become a true engineering discipline, many metrics and measurement approaches are needed:

- **Accurate Effort, Cost, and Schedule Data**
- **Accurate Defect, Quality, and User-Satisfaction Data**
- **Accurate Usage data**
- **Source code volumes for all languages**
- **Types and volumes of paper documents**
- **Volume of data and information used by software**
- **Consistent and reliable complexity information**

EVEN QUANTITATIVE DATA IS NOT ENOUGH!!

Reliable qualitative information is also needed. For every project, "soft" data should be collected:

- **Complete software process assessments**
- **Data on SEI capability maturity levels**
- **Extent of creeping requirements over entire life**
- **Methods, Tools, and Approaches Used**
- **Geographic factors**
- **Organizational Factors**
- **Experience levels of development team**
- **Project risks and value factors**

FUNCTION POINTS AND FUTURE METRICS

- **Software is only one technology used in modern business, in manufacturing, in government, and in military organizations.**
- **Can function points be expanded to other business activities?**
- **Can we build a suite of metrics similar to function points, for other aspects of modern business?**

FUNCTION POINTS AND OTHER METRICS

POTENTIAL BUSINESS METRICS

- **Function points -** **Measures software size**
- **Data points -** **Measures data base size**
- **Service points -** **Measures support size**
- **Engineering points -** **Measures hardware size**
- **Value points -** **Measures tangible value**

PROPOSED SUITE OF METRICS

Function Points
- Inputs
- Outputs
- Inquiries
- Logical files
- Interfaces

Service Points
- Customers
- Inputs
- Outputs
- Inquiries
- Logical files
- Interfaces
- Constraints
- References

Data Points
- Entities
- Sets
- Attributes
- Interfaces
- Constraints

Engineering Points
- Algorithms
- Inventions
- References
- Feedback
- Constraints
- Inputs
- Outputs
- Interfaces

Value Points
- Time to market
- Cost reduction
- Revenue increase
- Market share
- Morale
- Health/Safety
- Risk reduction
- National security
- Mandates/statutes
- Customer satisfaction
- Synergy
- Competitiveness
- Performance increases

POTENTIAL INTEGRATED COST ANALYSIS

Unit of Measure	Size	Unit $	Total Costs
Function points	1,000	$500	$500,000
Data points	2,000	$300	$600,000
Service points	1,500	$250	$375,000
Engineering points	1,500	$700	$1,050,000
TOTAL	6,000	$420	$2,525,000
Value points	10,000	$600	$6,000,000
ROI		$2.37 per $1.00 spent	

SUMMARY AND CONCLUSIONS

- **Function Points have aided software management but > 25 variations are excessive.**
- **Scientific and software engineering complexity should be included in function point analysis.**
- **The speed and cost of function point analysis need to be improved.**
- **Function points should aim for 100% usage between 1 function point and 1,000,000 function points!**
- **Function points should aim for usage analysis and portfolio analysis to become a major metric for global economic studies!**

SUMMARY AND CONCLUSIONS

- **Software is a major business tool for all industries, all levels of government, and all military organizations.**

- **Function points are the only metric that can measure quality, productivity, costs, value, and economics without distortion.**

- **Function points can become a key tool for large-scale economic analysis in every industry and country in the world!**

- **Function points can also become a key factor in software risk analysis and risk avoidance. No other metric can do this.**

- **Function points can improve the professional status of software and turn "software engineering" into a true engineering discipline instead of a craft as it is today.**

REFERENCES TO FUNCTION POINT MEASUREMENTS

- Garmus, David & Herron David, Function Point Analysis, Addison Wesley, 2001.
- IFPUG; IT Measurement; Addison Wesley, 2002
- Jones, Capers; Applied Software Measurement; McGraw Hill, 2008.
- Jones, Capers; Assessments, Benchmarks, and Best Practices; Addison Wesley, 2000.
- Jones, Capers; Estimating Software Costs; McGraw Hill,2007.
- Kan, Steve; Metrics and Models in Software Quality Engineering, Addison Wesley, 2003.
- Pressman, Roger; Software Engineering – A Practitioners Approach; McGraw Hill, 2005.
- Putnam, Larry; Measures for Excellence; Yourdon Press, Prentice Hall, 1992
- Symons, Charles; Software Sizing and Estimating (MK II), John Wiley & Sons 1992.
- Web sites:

• ITMPI.ORG	COSMIC.ORG	Relativity.COM	CSPIN.ORG
• IFPUG.ORG	ISO.ORG	ISBSG.ORG	FISMA.FI
• SPR.COM	ISMA.ORG	IFPUG.gr.jp	GUFPI.ORG