Function Points as a Universal Software Metric

Capers Jones, VP and CTO **Namcook Analytics LLC**

Draft 10.0 July 13, 2013

Blog: http://Namcookanalytics.com; Web: WWW.Namcook.com

Keywords

Capers Jones data, function point metrics, software benchmarks, software measurement, software metrics, software productivity, software quality.

Abstract

Function point metrics are the most accurate and effective metrics yet developed for software sizing and also for studying software productivity, quality, costs, risks, and economic value.

In the future function point metrics can easily become a universal metric used for all software applications and for all software contracts in all countries. However, there are some logistical problems with function point metrics that need to be understood and overcome in order for function point metrics to become the primary metric for software economic analysis.

Manual function point counting is too slow and costly to be used on large software projects above 10,000 function points in size. Also, application size is not constant but grows at about 2% per calendar month during development and 8% or more per calendar year for as long as software is in active use.

This paper discusses a method of high-speed function point counting that can size any application in less than two minutes, and which can predict application growth during development and for five years after release. This new method is based on pattern matching and is covered by U.S. utility patent application and hence is patent pending.

Table of Contents

Introduction

The main software cost drivers in descending order of magnitude include:

1. Finding and fixing defects
2. Producing paper documents
3. Coding or programming
4. Meetings and communications
5. Management
6. Dealing with software requirements changes

Function point metrics are the only available metric that can measure all six of the cost drivers individually, or together for economic analysis of total software projects. The older "lines of code" or LOC metric only measured coding, and did not even measure coding accurately. LOC metrics penalize modern high-level programming languages as discussed later in topic 14 of this article.

CEO's and other C-level executives want to know much more than just the coding part of software applications. They want to know the full cost of applications and their complete schedules from requirements through delivery. They also want to know multi-year maintenance and enhancement costs plus total cost of ownership (TCO).

Function point metrics were invented by A.J. Albrecht and colleagues at IBM's White Plains development center circa 1975. Function point metrics were placed in the public domain by IBM in 1978. Responsibility for function point counting rules soon transferred to the International Function Point User's Group (IFPUG). Their web site is www.IFPUG.org.

Function point metrics were developed by IBM due to serious mathematical and economic problems associated with the older "lines of code" metric or LOC. The LOC metric penalizes high-level programming languages and also cannot be used to evaluate requirements, design, business analysis, user documentation, or any other non-coding activities.

In the current era circa 2013 function point metrics are the major metric for software economic and productivity studies. At least 50,000 software projects have been measured using IFPUG function point metrics, including more than 5,000 projects that are publically available from the International Software Benchmark Standards Group (ISBSG). Their web site is www.ISBSG.org. See also the author's blog or http://Namcookanalytics.com.

The Strengths of Function Point Metrics

1. IFPUG function point metrics have more measured projects than all other metrics combined.
2. IFPUG function point metrics are endorsed by ISO/IEC standard 20926:2009.
3. Formal training and certification examinations are available for IFPUG function point counting.
4. Hundreds of certified IFPUG function point counters are available in most countries.
5. Counts of function points by certified counters usually are within 5% of each other.
6. IFPUG function point metrics are standard features of most parametric estimating tools such as KnowledgePlan, SEER, and Software Risk Master.
7. Function points are increasingly used for software contracts. The government of Brazil requires function points for all software contracts.

The Weaknesses of Function Point Metrics

1. Function point analysis is slow. Counting speeds for function points average perhaps 500 function points per day.
2. Due to the slow speed of function point analysis, function points are almost never used on large systems > 10,000 function points in size.
3. Function point analysis is expensive. Assuming a daily counting speed of 500 function points and a daily consulting fee of $1,500 counting an application of 10,000 function points would require 20 days and cost $30,000. This is equal to a cost of $3.00 for every function point counted.
4. Application size is not constant. During development applications grow at perhaps 2% per calendar month. After development applications continue to grow at perhaps 8% per calendar year. Current counting rules do not include continuous growth.
5. More than a dozen function point counting variations exist circa 2013 including COSMIC function points, NESMA function points, FISMA function points, fast function points, backfired function points, and a number of others. These variations produce function point totals that differ from IFPUG function points by perhaps + or – 15%.

A Patented Method for High-Speed Function Point Analysis

In order to make function point metrics easier to use and more rapid, the author filed a U.S. utility patent application in 2012. The utility patent application is U.S. Patent Application No. 13/352,434 filed January 18, 2012.

The high-speed sizing method is embedded in the Software Risk Master ™ (SRM) sizing and estimating tool under development by Namcook Analytics LLC. A working version is available on the Namcook Analytics web site, www.Namcook.com. The version requires a password from within the site.

The Namcook Analytics high-speed method includes these features:

1. From more than 200 trials sizing speed averages about 1.87 minutes per application. This speed is more or less constant between applications as small as 10 function points or as large as 300,000 function points.
2. The sizing method often comes within 5% of manual counts by certified counters. The closest match was an SRM predicted size of 1,802 function points for an application sized manually at 1,800 function points.
3. The sizing method can also be used prior to full requirements, which is the earliest of any known software sizing method.
4. The patent-pending method is based on external pattern matching rather than internal attributes. So long as an application can be placed on the SRM taxonomy the application can be sized.
5. The method can size all types of software including operating systems, ERP packages, telephone switching systems, medical device software, web applications, smart-phone applications, and normal information systems applications.
6. The sizing method is metric neutral and predicts application size in a total of 15 metrics including IFPUG function points, the new SNAP metric for non-functional attributes; COSMIC function points, story points, use-case points, logical code statements, and many others.
7. The sizing method predicts application growth during development and for five years of post-release usage.

A Short Summary of Pattern Matching

Today in 2013 very few applications are truly new. Most are replacements for older legacy applications or enhancements to older legacy applications. Pattern matching uses the size, cost, schedules, and other factors from legacy applications to generate similar values for new applications.

Software pattern matching as described here is based on a proprietary taxonomy developed by the author, Capers Jones. The taxonomy uses multiple-choice questions to identify the key attributes of software projects. The taxonomy is used to collect historical benchmark data and also as basis for estimating future projects. The taxonomy is also used for sizing applications.

For sizing, the taxonomy includes project nature, scope, class, type, problem complexity, code complexity, and data complexity. For estimating, additional parameters such as CMMI level, methodology, and team experience are also used.

The proprietary Namcook taxonomy used for pattern matching contains 122 factors. With 122 total elements the permutations of the full taxonomy total to 214,200,000 possible patterns. Needless to say more than half of these patterns have never occurred and will never occur.

For the software industry in 2013 the total number of patterns that occur with relatively high frequency is much smaller: about 20,000.

The Software Risk Master tool uses the taxonomy to select similar projects from its knowledge base of around 15,000 projects. Mathematical algorithms are used to derive results for patterns that do not have a perfect match.

However a great majority of software projects do have matches because they have been done many times. For example all banks perform similar transactions for customers and therefore have similar software packages. Telephone switches also have been done many times and all have similar features.

Pattern matching with a good taxonomy to guide the search is a very cost-effective way for dealing with application size.

Pattern matching is new for software sizing but common elsewhere. Two examples of pattern matching are the Zillow data base of real-estate costs and the Kelley Blue Book of used automobile costs. Both use taxonomies to narrow down choices, and then show clients the end results of those choices.

Increasing Executive Awareness of Function Points for Economic Studies

Because of the slow speed of function point analysis and the lack of data from large applications function points are a niche metric below the interest level of most CEO's and especially CEO's of Fortune 500 companies with large portfolios and many large systems including ERP packages.

In order for function point metrics to become a priority for C level executives and a standard method for all software contracts, some improvements are needed:

1. Function point size must be available in a few minutes for large systems; not after weeks of counting.
2. The cost per function point counted must be lower than $0.05 per function point rather than today's costs of more than $3.00 per function point counted.
3. Function point metrics must be able to size applications ranging from a low of 1 function point to a high of more than 300,000 function points.
4. Sizing of applications must also deal with the measured rates of requirements creep during development and the measured rates of post-release growth for perhaps 10 years after the initial release.
5. Function points must also be applied to maintenance, enhancements, and total costs of ownership (TCO).
6. Individual changes in requirements should be sized in real-time as they occur. If a client wants a new feature that may be 10 function points in size, this fact should be established within a few minutes.
7. Function points should be used for large-scale economic analysis of methods, industries, and even countries.
8. Function points should be used to measure consumption and usage of software as well as production of software.
9. Function points must be applied to portfolio analysis and system architecture.
10. Function points can and should be used to size and estimate collateral materials such as documentation, test case volumes, and reusable materials used for applications.

Topic 1: Sizing Application Growth during Development and After Release

Software application size is not a constant. Software projects grow continuously during development and also after release. Following are some examples of the features of the patent-pending sizing method embedded in Software Risk Master ™. Table 1 shows an example of the way SRM predicts growth and post-release changes:

Table 1: Software Risk Master™ Multi-Year Sizing

Patent application 13/352,434.

	Nominal application size in IFPUG function points	**10,000**
		Function Points
1	Size at end of requirements	10,000
2	Size of requirement creep	2,000
3	Size of planned delivery	12,000
4	Size of deferred functions	-4,800
5	Size of actual delivery	7,200
6	Year 1	12,000
7	Year 2	13,000
8	Year 3	14,000
9	**Year 4***	**17,000**
10	Year 5	18,000
11	Year 6	19,000
12	Year 7	20,000
13	**Year 8***	**23,000**
14	Year 9	24,000
15	Year 10	25,000

Note that calendar years 4 and 8 show a phenomenon called "mid-life kickers" or major new features added about every four years to commercial software applications. Multi-year sizing is based on empirical data from a number of major companies such as IBM where applications have been in service for more than 10 years.

Software applications should be resized whenever there are major enhancements. Individual enhancements should be sized and all data should be accumulated starting with the initial delivery. Formal sizing should also take place at the end of every calendar year for applications that are deployed and in active use.

Topic 2: Predicting Application Size in Multiple Metrics

There are so many metrics in use in 2013 that as a professional courtesy to users and other metrics groups SRM predicts size in the metrics shown in Table 2. Assume that the application being sized is known to be 10,000 function points using IFPUG version 4.2 counting rules:

Table 2: Metrics Supported by SRM Pattern Matching

Alternate Metrics	Size	% of IFPUG
Backfired function points	10,000	100.00%
Cosmic function points	11,429	114.29%
Fast function points	9,700	97.00%
Feature points	10,000	100.00%
FISMA function points	10,200	102.00%
Full function points	11,700	117.00%
Function points light	9,650	96.50%
Mark II function points	10,600	106.00%
NESMA function points	10,400	104.00%
RICE objects	47,143	471.43%
SCCQI "function points"	30,286	302.86%
SNAP non functional metrics	1,818	18.18%
Story points	5,556	55.56%
Unadjusted function points	8,900	89.00%
Use case points	3,333	33.33%

Because SRM is metric neutral, additional metrics could be added to the list of supported metrics if new metrics become available in the future.

SRM also predicts application size in terms of logical code statements or "LOC." However with more than 2,500 programming languages in existence and the majority of projects using several languages, code sizing requires that users inform the SRM tool as to which language(s) will be used. This is done by specifying a percentage of various languages from an SRM pull-down menu that lists the languages supported. Currently SRM supports about 180 languages for sizing, but this is just an arbitrary number that can easily be expanded.

Topic 3: Sizing All Known Types of Software Application

One of the advantages of sizing by means of external pattern matching rather than sizing by internal attributes is that the any known application can be sized. Table 3 shows 40 samples of applications size by the Software Risk Master ™ patent-pending method:

Table 3: Examples of Software Size via Pattern Matching

Using Software Risk Master ™

Application	**Size in IFPUG Function Points**
1. Oracle	229,434
2. Windows 7 (all features)	202,150
3. Microsoft Windows XP	66,238
4. Google docs	47,668
5. Microsoft Office 2003	33,736
6. F15 avionics/weapons	23,109
7. VA medical records	19,819
8. Apple I Phone	19,366
9. IBM IMS data base	18,558
10. Google search engine	18,640
11. Linux	17,505
12. ITT System 12 switching	17,002
13. Denver Airport luggage (original)	16,661
14. Child Support Payments (state)	12,546
15. Facebook	8,404
16. MapQuest	3,793
17. Microsoft Project	1,963
18. Android OS (original version)	1,858
19. Microsoft Excel	1,578
20. Garmin GPS navigation (hand held)	1,518
21. Microsoft Word	1,431
22. Mozilla Firefox	1,342
23. Laser printer driver (HP)	1,248
24. Sun Java compiler	1,185
25. Wikipedia	1,142
26. Cochlear implant (embedded)	1,041
27. Microsoft DOS circa 1998	1,022
28. Nintendo Gameboy DS	1,002
29. Casio atomic watch	933
30. Computer BIOS	857
31. SPR KnowledgePlan	883
32. Function Point Workbench	714
33. Norton anti-virus	700
34. SPR SPQR/20	699

35.	Golf handicap analysis	662
36.	Google Gmail	590
37.	Twitter (original circa 2009)	541
38.	Freecell computer solitaire	102
39.	Software Risk Master™ prototype	38
40.	ILOVEYOU computer worm	22

This list of 40 applications was sized by the author in about 75 minutes, which is a rate of 1.875 minutes per application sized. The cost per function point sized is less than $0.001. As of 2013 SRM sizing is the fastest and least expensive method of sizing yet developed. This makes SRM useful for Agile projects where normal function point analysis is seldom used.

Because sizing is based on external attributes rather than internal factors, SRM sizing can take place before full requirements are available; this is 3 to 6 months earlier than most other sizing methods. Early sizing leaves time for risk abatement for potentially hazardous projects.

Topic 4: Function Points for Early Analysis of Software Risks

Software projects are susceptible to more than 200 risks in all, of which about 50 can be analyzed using function point metrics. As application size goes up when measured with function point metrics, software risks also go up.

Table 4 shows the comparative risk profiles of four sample projects of 100, 1000, 10,000, and 100,000 function points. All four are "average" projects using iterative development. All four are assumed to be at CMMI level 1.

Table 4: Average Risks for IT Projects by Size

(Predictions by Software Risk Master ™)

Risks for 100 function points	
Cancellation	**8.36%**
Negative ROI	**10.59%**
Cost overrun	**9.19%**
Schedule slip	**11.14%**
Unhappy customers	**36.00%**
Litigation	**3.68%**
Average Risks	**13.16%**
Financial Risks	**15.43%**
Risks for 1000 function points	
Cancellation	**13.78%**
Negative ROI	**17.46%**
Cost overrun	**15.16%**
Schedule slip	**18.38%**
Unhappy customers	**36.00%**
Litigation	**6.06%**
Average Risks	**17.81%**
Financial Risks	**25.44%**
Risks for 10,000 function points	
Cancellation	**26.03%**
Negative ROI	**32.97%**
Cost overrun	**28.63%**
Schedule slip	**34.70%**
Unhappy customers	**36.00%**
Litigation	**11.45%**
Average Risks	**28.29%**
Financial Risks	**48.04%**

Risks for 100,000 function points	
Cancellation	**53.76%**
Negative ROI	**68.09%**
Cost overrun	**59.13%**
Schedule slip	**71.68%**
Unhappy customers	**36.00%**
Litigation	**23.65%**
Average Risks	**52.05%**
Financial Risks	**99.24%**

All of the data in Table 4 are standard risk predictions from Software Risk Master ™. Risks would go down with higher CMMI levels, more experienced teams, and robust methodologies such a RUP or TSP.

Small projects below 1000 function points are usually completed without too much difficulty. But large systems above 10,000 function points are among the most hazardous of all manufactured objects in human history.

It is an interesting phenomenon that every software breach of contract lawsuit except one where the author worked as an expert witness were for projects of 10,000 function points and higher.

Topic 5: Function Points for Activity-Based Sizing and Cost Estimating

In order to be useful for software economic analysis, function point metrics need to be applied to individual software development activities. Corporate executives at the CEO level want to know all cost elements, and not just "design, code, and unit test" or DCUT as it is commonly called.

SRM has a variable focus that allows it to show data ranging from full projects to 40 activities. Table 5 shows the complete set of 40 activities for an application of 10,000 function points in size:

Table 5: Function Points for Activity-Based Cost Analysis

	Development Activities	Work Hours per Function Point	Burdened Cost per Function Point
1	Business analysis	0.02	$1.33
2	Risk analysis/sizing	0.00	$0.29
3	Risk solution planning	0.01	$0.67
4	Requirements	0.38	$28.57
5	Requirement. Inspection	0.22	$16.67
6	Prototyping	0.33	$25.00
7	Architecture	0.05	$4.00
8	Architecture. Inspection	0.04	$3.33
9	Project plans/estimates	0.03	$2.00
10	Initial Design	0.75	$57.14
11	Detail Design	0.75	$57.14
12	Design inspections	0.53	$40.00
13	Coding	4.00	$303.03
14	Code inspections	3.30	$250.00
15	Reuse acquisition	0.01	$1.00
16	Static analysis	0.02	$1.33
17	COTS Package purchase	0.01	$1.00
18	Open-source acquisition.	0.01	$1.00
19	Code security audit.	0.04	$2.86
20	Ind. Verif. & Valid.	0.07	$5.00
21	Configuration control.	0.04	$2.86
22	Integration	0.04	$2.86
23	User documentation	0.29	$22.22
24	Unit testing	0.88	$66.67
25	Function testing	0.75	$57.14
26	Regression testing	0.53	$40.00
27	Integration testing	0.44	$33.33

28	Performance testing	0.33	$25.00
29	Security testing	0.26	$20.00
30	Usability testing	0.22	$16.67
31	System testing	0.88	$66.67
32	Cloud testing	0.13	$10.00
33	Field (Beta) testing	0.18	$13.33
34	Acceptance testing	0.05	$4.00
35	Independent testing	0.07	$5.00
36	Quality assurance	0.18	$13.33
37	Installation/training	0.04	$2.86
38	Project measurement	0.01	$1.00
39	Project office	0.18	$13.33
40	Project management	4.40	$333.33
	Cumulative Results	20.44	$1,548.68

SRM uses this level of detail for collecting benchmark data from large applications. In predictive mode prior to requirements this much detail is not needed, so a smaller chart of accounts is used.

This chart of accounts works for methods such as waterfall. Other charts of accounts are used for iterative, agile, and other methods which segment development into sprints or separate work packages.

Major applications that have separate components would use a chart of accounts for each component. All of these could be merged at the end of the project.

One advantage of activity-based costing with function point metrics is that it eliminates "leakage" from measurement studies. For too many studies cover only "design, code, and unit test" or DCUT. These partial activities are less than 30% of the effort on a large software application.

C-level executives want and should see 100% of the total set of activities that goes into software projects, not just partial data. From analysis of internal measurement programs IT projects only collect data on about 37% of the total effort for software. Only contract software with charges on a time and material basis approach 100% cost collection, plus defense applications developed under contract.

A few major companies such as IBM collect data from internal applications that approach 100% in completeness, but this is fairly rare. The most common omissions are unpaid overtime, project management, and the work of part-time specialists such as business analysts, quality assurance, technical writers, function point counters, and the like.

Topic 6: Function Points and Methodology Analysis

One topic of considerable interest to both C level executives and also to academics and software engineers is how various methodologies compare. Software Risk Master ™ includes empirical results from more than 30 different software development methodologies; more than any other benchmark or estimation tool.

Table 6 shows the approximate development schedules noted for 30 different software development methods. The rankings run from slowest at the top of the table to fastest at the bottom of the table:

Table 6: Application Schedules

	Application Size = (IFPUG 4.2)	**10**	**100**	**1,000**	**10,000**	**100,000**
	Methods	**Schedule Months**	**Schedule Months**	**Schedule Months**	**Schedule Months**	**Schedule Months**
1	Proofs	2.60	6.76	17.58	45.71	118.85
2	DoD	2.57	6.61	16.98	43.65	112.20
3	Cowboy	2.51	6.31	15.85	39.81	100.00
4	Waterfall	2.48	6.17	15.31	38.02	94.41
5	ISO/IEC	2.47	6.11	15.10	37.33	92.26
6	Pairs	2.45	6.03	14.79	36.31	89.13
7	Prince2	2.44	5.94	14.49	35.32	86.10
8	Merise	2.44	5.94	14.49	35.32	86.10
9	DSDM	2.43	5.92	14.39	34.99	85.11
10	Models	2.43	5.89	14.29	34.67	84.14
11	Clean rm.	2.42	5.86	14.19	34.36	83.18
12	T-VEC	2.42	5.86	14.19	34.36	83.18
13	V-Model	2.42	5.83	14.09	34.04	82.22
14	Iterative	2.41	5.81	14.00	33.73	81.28
15	SSADM	2.40	5.78	13.90	33.42	80.35
16	Spiral	2.40	5.75	13.80	33.11	79.43
17	SADT	2.39	5.73	13.71	32.81	78.52
18	Jackson	2.39	5.73	13.71	32.81	78.52
19	EVO	2.39	5.70	13.61	32.51	77.62
20	IE	2.39	5.70	13.61	32.51	77.62
21	OO	2.38	5.68	13.52	32.21	76.74
22	DSD	2.38	5.65	13.43	31.92	75.86
23	RUP	2.37	5.62	13.34	31.62	74.99
24	PSP/TSP	2.36	5.56	13.11	30.90	72.86

25	FDD	2.36	5.55	13.06	30.76	72.44
26	RAD	2.35	5.52	12.97	30.48	71.61
27	Agile	2.34	5.50	12.88	30.20	70.79
28	XP	2.34	5.47	12.79	29.92	69.98
29	Hybrid	2.32	5.40	12.53	29.11	67.61
30	Mashup	2.24	5.01	11.22	25.12	56.23
	Average	**2.41**	**5.81**	**14.03**	**33.90**	**81.98**

Software Risk Master ™ also predicts staffing, effort in months and hours, costs, quality, and 5-years of post-release maintenance and enhancement. Table 6 only shows schedules since that topic is of considerable interest to CEO's as well as other C level executives.

Note that table 6 assumes close to a zero value for certified reusable components. Software reuse can shorten schedules compared those shown in table 6.

Table 6 also assumes an average team and no use of the capability maturity model. Expert teams and projects in organizations at CMMI levels 3 or higher will have shorter schedules than those shown in table 6.

SRM itself handles adjustments in team skills, CMMI levels, methodologies, programming languages, and volumes of reuse.

Topic 7: Function Points for Evaluating the Capability Maturity Model (CMMI ®)

Civilian CEO's in many industries such as banking, insurance, commercial software, transportation, and many kinds of manufacturing care little or nothing about the Software Engineering Institute's (SEI) capability maturity model. In fact many have never even heard of either the SEI or the CMMI.

In the defense industry, on the other hand, the CMMI is a major topic because it is necessary to be at CMMI level 3 of higher in order to bid on some military and defense software contracts. The SEI was formed in 1984 and soon after started doing software process assessments, using a methodology developed by the late Watts Humphrey and his colleagues.

The original assessment method scored organizations on a 5-point scale ranging from 1 (very chaotic in software) through 5 (highly professional and disciplined). The author of this report had a contract with the U.S. Air Force to explore the value of ascending the various CMM and CMMI levels to ascertain if there were tangible improvements in quality and productivity. Twenty companies were visited for the study.

There are tangible improvements in both quality and productivity gains at the higher CMMI levels from a statistical analysis. However some companies that don't use the CMMI at all have results as good as companies assessed at CMMI level 5. Tables 7.1 through 7.4 show CMMI results for various sizes of projects:

Table 7.1: Quality Results of the Five Levels of the CMMI ®
(Results for applications of 1000 function points in size)

CMMI Level	Defect Potential per FP	Defect Removal Efficiency	Delivered Defects per FP
1	5.25	82.00%	0.95
2	5.00	85.00%	0.75
3	4.75	90.00%	0.48
4	4.50	94.00%	0.27
5	4.00	98.00%	0.08

Table 7.2: Quality Results of the Five Levels of the CMMI ®
(Results for applications of 10,000 function points in size)

CMMI Level	Defect Potential per FP	Defect Removal Efficiency	Delivered Defects per FP
1	6.50	75.00%	1.63
2	6.25	82.00%	1.13
3	5.50	87.00%	0.72
4	5.25	90.00%	0.53
5	4.50	94.00%	0.27

Table 7.3: Productivity Rates for the Five Levels of the CMMI ®
(Results for applications of 1000 function points in size)

CMMI Level	Function Points per Month	Work Hours per Function Point
1	6.00	22.00
2	6.75	19.56
3	7.00	18.86
4	7.25	18.21
5	7.50	17.60

Table 7.4: Productivity Rates for the Five Levels of the CMMI ®
(Results for applications of 10,000 function points in size)

CMMI Level	Function Points per Month	Work Hours per Function Point
1	2.50	52.80
2	2.75	48.00
3	3.50	37.71
4	4.25	31.06
5	5.50	24.00

The higher levels of the CMMI have better quality than similar civilian projects, but much lower productivity. This is due in part to the fact the DoD oversight leads to huge volumes of paper documents. Military projects create about three times as many pages of requirements, specifications, and other documents as do civilian projects of the same size. Software paperwork costs more than source code for military and defense applications regardless of CMMI levels. DoD projects also use independent verification and validation (IV&V) and independent testing, which seldom occurs in the civilian sector.

Both Namcook Analytics and the SEI express their results using five-point scales but the significance of the scales runs in the opposite direction. The Namcook scale was published first in 1986 in Capers Jones' book Programming Productivity (McGraw Hill) and hence is several years older than the SEI scale. In fact the author began doing assessments inside IBM at the same time Watts Humphrey was IBM's director of programming, and about 10 years before the SEI was even incorporated.

Namcook Excellence Scale	**Meaning**	**Frequency of Occurrence**
1 = Excellent	State of the art	2.0%
2 = Good	Superior to most companies	18.0%
3 = Average	**Normal in most factors**	**56.0%**
4 = Poor	Deficient in some factors	20.0%
5 = Very Poor	Deficient in most factors	4.0%

The SEI maturity level scale was first published by Watts Humphrey in 1989 in his well-known book Managing the Software Process (Humphrey 1989):

SEI Maturity Level	**Meaning**	**Frequency of Occurrence**
1 = Initial	Chaotic	75.0%
2 = Repeatable	Marginal	15.0%
3 = Defined	Adequate	8.0%
4 = Managed	Good to excellent	1.5%
5 = Optimizing	State of the art	0.5%

Simply inverting the Namcook excellence scale or the SEI maturity scale is not sufficient to convert the scores from one to another. This is because the SEI scale expresses its results in absolute form, while the Namcook scale expresses its results in relative form.

Large collections of Namcook data from an industry typically approximate a bell-shaped curve. A collection of SEI capability maturity data is skewed toward the Initial or chaotic end of the

spectrum. However, it is possible to convert data from the Namcook scale to the equivalent SEI scale by a combination of inversion and compression of the Namcook results. Software Risk Master ™ includes bi-directional conversions between the SEI and Namcook scales.

The Namcook conversion algorithms use two decimal places, so that scores such as 3.25 or 3.75 are possible. In fact even scores below 1 such as 0.65 are possible. The SEI method itself uses integer values but the two-decimal precision of the Namcook conversion method is interesting to clients.

Topic 8: Function Points for Software Quality Analysis

Function points are the best metric for software quality analysis. The older metric "cost per defect" penalizes quality and also violates standard economic assumptions. Quality economics are much better analyzed using function point metrics than any other. A key advantage of function points for quality analysis is the ability to predict defects in requirements and design as well as code defects. Requirements and design defects often outnumber code defects. Table 8 shows a typical defect pattern for a waterfall project of 1000 function points at CMMI level 1 coding in the C language:

Table 8: SRM Defect Prediction for Waterfall Development

1000 function points; inexperienced team; CMMI Level 1; Waterfall; C language
106,670 logical code statements; 106.7 KLOC

Defect Potentials	Defects	Per FP	Per KLOC	Pre-Test Removal	Test Removal	Defects Delivered	Cumul. Effic.
Requirements	1,065	1.07	9.99	70.00%	54.00%	**147**	86.20%
Design	1,426	1.43	13.37	76.00%	69.00%	**106**	92.56%
Code	1,515	1.52	14.20	77.00%	74.00%	**91**	94.02%
Documents	665	0.66	6.23	79.00%	19.00%	**113**	82.99%
Bad fixes	352	0.35	3.30	59.00%	61.00%	**56**	84.01%
TOTAL	5,023	5.02	47.09	74.24%	60.36%	**513**	**89.79 %**

Table 8.1 shows a sample of the full quality predictions from SRM for an application of 1000 function points. The table shows a "best case" example for a project using TSP and being at CMMI level 5:

Table 8.1: SRM Quality Estimate	**Output Results**
Requirements defect potential	**134**
Design defect potential	**561**
Code defect potential	**887**
Document defect potential	**135**
Total Defect Potential	**1,717**
Per function point	**1.72**
Per KLOC	**32.20**

Defect Prevention	Efficiency	Remainder	Bad Fixes	Costs
JAD	27%	**1,262**	**5**	$28,052
QFD	30%	**88**	**4**	$39,633

		871		
Prototype	20%	322	2	$17,045
Models	68%	923	5	$42,684
Subtotal	**86%**	4	15	$127,415

Pre-Test Removal	**Efficiency**	**Remainder**	**Bad Fixes**	**Costs**
		17		
Desk check	27%	1	2	$13,225
Static analysis	55%	78	1	$7,823
Inspections	93%	5	0	$73,791
Subtotal	**98%**	6	3	$94,839

Test Removal	**Efficiency**	**Remainder**	**Bad Fixes**	**Costs**
Unit	32%	4	0	$22,390
Function	35%	2	0	$39,835
Regression	14%	2	0	$51,578
Component	32%	1	0	$57,704
Performance	14%	1	0	$33,366
System	36%	1	0	$63,747
Acceptance	17%	1	0	$15,225
Subtotal	**87%**	1	0	$283,845

				Costs
PRE-RELEASE COSTS		1,734	3	**$506,099**
POST-RELEASE REPAIRS	**(TECHNICAL DEBT)**	1	0	**$658**
MAINTENANCE OVERHEAD				**$46,545**
COST OF QUALITY	**(COQ)**			**$553,302**

Defects delivered	1

High severity	**0**
Security flaws	**0**
High severity %	**11.58%**
Delivered Per FP	**0.001**
High severity per FP	**0.000**
Security flaws per FP	**0.000**
Delivered Per KLOC	**0.014**
High severity per KLOC	**0.002**
Security flaws per KLOC	**0.001**
Cumulative Removal Efficiency	**99.96%**

Function points are able to quantify requirements and design defects, which outnumber coding defects for large applications. This is not possible using LOC metrics. Function points are also superior to "cost per defect" for measuring technical debt and cost of quality (COQ). Both technical debt and COQ are standard SRM outputs.

Table 8.1 is only an example. SRM can also model various ISO standards, certification of test personnel, team experience levels, CMMI levels, and in fact a total of about 200 specific quality factors.

For many years the software industry has used "cost per defect" as key metric for software quality. In fact there is an urban legend that "it costs 100 times as much to fix a bug after release as early in development." Unfortunately the whole concept of cost per defect is mathematically flawed and does match standard economics. The urban legend has values that resemble the following:

Defects found during requirements =	$250
Defects found during design =	$500
Defects found during coding and testing =	$1,250
Defects found after release =	$5,000

While such claims are often true mathematically, there are three hidden problems with cost per defect that are usually not discussed in the software literature:

1. Cost per defect penalizes quality and is always cheapest where the greatest numbers of bugs are found.

2. Because more bugs are found at the beginning of development than at the end, the increase in cost per defect is artificial. Actual time and motion studies of defect repairs show little variance from end to end.

3. Even if calculated correctly, cost per defect does not measure the true economic value of improved software quality. Over and above the costs of finding and fixing bugs, high quality leads to shorter development schedules and overall reductions in development costs. These savings are not included in cost per defect calculations, so the metric understates the true value of quality by several hundred percent.

Let us consider the cost per defect problem areas using examples that illustrate the main points.

As quality improves, "cost per defect" rises sharply. The reason for this is that writing test cases and running them act like fixed costs. It is a well-known law of manufacturing economics that:

"If a manufacturing cycle includes a high proportion of fixed costs and there is a reduction in the number of units produced, the cost per unit will go up."

As an application moves through a full test cycle that includes unit test, function test, regression test, performance test, system test, and acceptance test the time required to write test cases and the time required to run test cases stays almost constant; but the number of defects found steadily decreases.

Table 8.2 shows the approximate costs for the three cost elements of preparation, execution, and repair for the test cycles just cited using the same rate of $75.75 per hour for all activities:

Table 8.2: Cost per Defect for Six Forms of Testing
(Assumes $75.75 per staff hour for costs)

	Writing Test Cases	Running Test Cases	Repairing Defects	TOTAL COSTS	Number of Defects	$ per Defect
Unit test	$1,250.00	$750.00	$18,937.50	$20,937.50	50	$418.75
Function test	$1,250.00	$750.00	$7,575.00	$9,575.00	20	$478.75
Regression test	$1,250.00	$750.00	$3,787.50	$5,787.50	10	$578.75
Performance test	$1,250.00	$750.00	$1,893.75	$3,893.75	5	$778.75
System test	$1,250.00	$750.00	$1,136.25	$3,136.25	3	$1,045.42
Acceptance test	$1,250.00	$750.00	$378.75	$2,378.75	1	$2,378.75

What is most interesting about table 8.2 is that cost per defect rises steadily as defect volumes come down, even though table 8.2 uses a constant value of 5 hours to repair defects for every single test stage! In other words every defect identified throughout table 1 had a constant cost of $378.25 when only repairs are considered.

In fact all three columns use constant values and the only true variable in the example is the number of defects found. In real life, of course, preparation, execution, and repairs would all be variables. But by making them constant, it is easier to illustrate the main point: ***cost per defect rises as numbers of defects decline.***

Let us now consider cost per function point as an alternative metric for measuring the costs of defect removal.

An alternate way of showing the economics of defect removal is to switch from "cost per defect" and use "defect removal cost per function point". Table 8.3 uses the same basic information as Table 8.3, but expresses all costs in terms of cost per function point:

Table 8.3 Cost per Function Point for Six Forms of Testing
(Assumes $75.75 per staff hour for costs)
(Assumes 100 function points in the application)

	Writing Test Cases	Running Test Cases	Repairing Defects	TOTAL $ PER F.P.	Number of Defects
Unit test	$12.50	$7.50	$189.38	$209.38	50
Function test	$12.50	$7.50	$75.75	$95.75	20
Regression test	$12.50	$7.50	$37.88	$57.88	10
Performance test	$12.50	$7.50	$18.94	$38.94	5
System test	$12.50	$7.50	$11.36	$31.36	3
Acceptance test	$12.50	$7.50	$3.79	$23.79	1

The advantage of defect removal cost per function point over cost per defect is that it actually matches the assumptions of standard economics. In other words, as quality improves and defect volumes decline, cost per function point tracks these benefits and also declines. High quality is shown to be cheaper than poor quality, while with cost per defect high quality is incorrectly shown as being more expensive.

Topic 9: Function Points and Software Maintenance, Enhancements, and Total Cost of Ownership (TCO)

Software costs do not end when the software is delivered. Nor does delivery put an end to the need to monitor both costs and quality. Some applications have useful lives that can span 20 years or more. These applications are not fixed, but add new features on an annual basis. Therefore function point metrics need to continue to be applied to software projects after release.

Post-release costs are more complex than development costs because they need to integrate enhancements or adding new features, maintenance or fixing bugs, and customer support or helping clients when they call or contact a company about a specific application.

The need to keep records for applications that are constantly growing over time means that normalization of data will need to be cognizant of the current size of the application. The method used by Software Risk Master ™ is to normalize results for both enhancements and maintenance at the end of every calendar year; i.e. the size of the application is based on the date of December 31. The pre-release size is based on the size of the application on the day it was first delivered to clients. The sizes of requirements creep during development are also recorded.

Table 9 shows the approximate rate of growth and the maintenance and enhancement effort for five years for an application of a nominal 1000 function points when first delivered:

Table 9: Five Years of Software Maintenance and Enhancement for 1000 Function Points

(MAINTENANCE + ENHANCEMENT)

	Year 1 2013	**Year 2 2014**	**Year 3 2015**	**Year 4 2016**	**Year 5 2017**	**5-Year Totals**
Annual enhancements in FP	80	86	93	101	109	**469**
Application Growth in FP	1,080	1,166	1,260	1,360	1,469	**1,469**
Application Growth in LOC	57,600	62,208	67,185	67,185	78,364	**78,364**
Cyclomatic complexity increase	11.09	11.54	12.00	12.48	12.98	**12.98**
Enhancement staff	0.81	0.88	0.96	1.05	1.15	0.97
Maintenance staff	5.68	5.72	5.85	6.36	7.28	6.18
Total staff	6.49	6.61	6.81	7.41	8.43	7.15
Enhancement effort (months)	9.72	10.61	11.58	12.64	13.80	58.34
Maintenance effort (months)	68.19	68.70	70.20	76.31	87.34	370.74
Total effort (months)	77.91	79.30	81.78	88.95	101.14	429.08
Total effort (hours)	10,283.53	10,467.77	10,794.70	11,741.94	13,350.37	56,638.31
Enhancement Effort %	12.47%	13.37%	14.16%	14.21%	13.64%	13.60%

Maintenance Effort %	87.53%	86.63%	85.84%	85.79%	86.36%	86.40%
Total Effort %	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Enhancement cost	$77,733	$84,845	$92,617	$101,114	$110,403	$466,712
Maintenance cost	$331,052	$316,674	$304,368	$315,546	$347,348	$1,614,988
Total cost	$408,785	$401,518	$396,985	$416,660	$457,751	$2,081,700
Enhancement cost %	19.02%	21.13%	23.33%	24.27%	24.12%	22.42%
Maintenance cost %	80.98%	78.87%	76.67%	75.73%	75.88%	77.58%
Total Cost	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

The original development cost for the application was $1,027,348. The costs for five years of maintenance and enhancements the cost were $2,081,700 or more than twice the original development cost. The total cost of ownership is the sum of development and the 5-year M&E period. In this example the TCO is $3,109,048.

CEO's and other C level executives want to know the "total cost of ownership" (TCO) of software and not just the initial development costs.

Five-year maintenance and enhancement predictions are standard outputs from Software Risk Master ™ (SRM).

High quality can reduce development costs by about 15% but reduces maintenance costs by more than 45% per year. The cumulative economic value of quality methods such as inspections and static analysis are much better when demonstrated using total cost of ownership (TCO) than when only using development.

The TCO of the "average" sample shown here was just over $3,000,000. The TCO of a high-quality version of the same application that used pre-test inspections and static analysis would probably have been below $1,500,000 with the bulk of the saving accruing after release due to lower customer support and maintenance costs.

Topic 10: Function Points and Forensic Analysis of Canceled Projects

CEO's of Fortune 500 companies have less respect for their software organizations than for other technical groups. The main reason for this is that large software projects are more troublesome to CEO's and corporate boards than any other manufactured object in history.

Large software projects are canceled far too often and run late and miss budgets in a majority of cases. Until quality improves along with estimation and safe completion of large systems, software organizations will be viewed as a painful necessity rather than a valuable contributor to the bottom line.

About 35% of large systems > 10,000 function points are canceled and never delivered to end users or clients. These canceled projects are seldom studied but forensic analysis of failures can lead to important insights.

The reason that CEO's care about forensic analysis is that many failed projects end up in litigation. Between the financial loss of the cancellation and possible legal fees for litigation, a failed project of 10,000 function points is about a $30,000,000 write off. A failure for a major system of 100,000 function points is about a $500,000,000 write off. This much wasted money is a top issue for CEO's.

Because they are terminated, the size of the project at termination and accumulated costs and resource data may not be available. But Software Risk Master ™ can provide both values.

Note that terminated projects have no downstream "technical debt" because they are never released to customers. This is a serious omission from the technical debt metaphor, and is one of the reasons why it is not completely valid for economic analysis.

Tables 10.1 through 10.3 are taken from Chapter 7 of the author's book The Economics of Software Quality, Addison Wesley, 2011.

Table 10.1: Odds of Cancellation by Size and Quality Level
(Includes negative ROI, poor quality, and change in business need)

Function Points	Low Quality	Average Quality	High Quality
10	2.00%	0.00%	0.00%
100	7.00%	3.00%	2.00%
1000	20.00%	10.00%	5.00%
10000	45.00%	15.00%	7.00%
100000	65.00%	35.00%	12.00%
Average	27.80%	12.60%	5.20%

Table 10.2: Probable Month of Cancellation from Start of Project
(Elapsed months from start of project)

Function Points	Low Quality	Average Quality	High Quality
10	1.4	None	None
100	5.9	5.2	3.8
1000	16.0	13.8	9.3
10000	38.2	32.1	16.1
100000	82.3	70.1	25.2
Average	45.5	30.3	16.9
Percent	150.10%	100.00%	55.68%

Table 10.3: Probable Effort from Project Start to Point of Cancellation
(Effort in terms of Person Months)

Function Points	Low Quality	Average Quality	High Quality
10	0.8	None	None
100	10.0	7.9	5.4
1000	120.5	92.0	57.0
10000	2,866.5	2,110.1	913.2
100000	61,194.7	45,545.5	13,745.5
Average	21,393.9	15,915.9	4,905.2
Percent	134.42%	100.00%	30.82%

When high-quality projects are canceled it is usually because of business reasons. For example the author was working on an application when his company bought a competitor that already had the same kind of application up and running. The company did not need two identical applications, so the version under development was canceled. This was a rational business decision and not due to poor quality or negative ROI.

When low-quality projects are canceled it is usually because they are so late and so much over budget that their return on investment (ROI) turned from positive to strongly negative. The delays and cost overruns explain why low-quality canceled projects are much more expensive

than successful projects of the same size and type. Function point metrics are the best choice for forensic analysis of canceled projects.

Much of the forensic analysis of disastrous projects takes place in the discovery and deposition phases of software litigation. From working as an expert witness in many of these cases, the main reasons for canceled projects include:

1. Optimistic estimates which predict shorter schedules and lower costs than reality.
2. Poor quality control which stretches out test schedules.
3. Poor change control in the face of requirements creep which tops 2% per calendar month.
4. Poor and seemingly fraudulent status tracking which conceals serious problems.

In general poor management practices are the chief culprit for canceled projects. Managers don't understand quality and try and bypass pre-test inspections or static analysis, which later doubles testing durations.

Then managers conceal problems from clients and higher management in the false hope that the problems will be solved or go away. The inevitable results are massive cost overruns, long schedule delays, and outright cancellation. It is no wonder CEO's have a low regard for the software groups in their companies.

Topic 11: Portfolio Analysis with Function Point Metrics

To be useful and interesting to CEO's and other C level executives function points should be able to quantify not just individual projects but also large collections of related projects such as the full portfolio of a Fortune 500 company.

A full corporate software portfolio is an expensive asset that needs accurate financial data. In fact the author has worked on several IRS tax cases involving the asset value of corporate portfolios. One of these tax cases involved the value of the EDS portfolio at the time it was acquired by General Motors. Another interesting tax case involved the asset value of the Charles Schwab software portfolio.

Since portfolios are taxable assets when companies are sold, it is obvious why they need full quantification. Function points are the best metric for portfolio quantification, although clearly faster methods are needed than manual function point analysis. Up until recently "backfiring" or mathematical conversion from source code to function points was the main method used for portfolio analysis.

Table 11 is an example of the special SRM predictions for corporate portfolios. This prediction shows the size in function points for the portfolio of a Fortune 500 manufacturing company with 100,000 total employees:

Table 11: Portfolio Analysis of a Fortune 500 Manufacturing Company

	Corporate Functions	Number of Applications Used	Function Points	Lines of Code
1	Accounts payable	18	26,674	1,467,081
2	Accounts receivable	22	33,581	1,846,945
3	Advertising	32	47,434	2,134,537
4	Advisory boards - technical	6	8,435	463,932
5	Banking relationships	38	131,543	7,234,870
6	Board of directors	4	6,325	347,900
7	Building maintenance	2	3,557	195,638
8	Business intelligence	21	73,972	4,068,466
9	Business partnerships	18	44,457	2,445,134
10	Competitive analysis	30	74,635	4,104,901
11	Consultant management	3	4,609	253,486

12	Contract management	32	94,868	5,217,758
13	Customer resource management	56	140,585	7,732,193
14	Customer support	45	67,003	3,685,140
15	Divestitures	10	15,000	825,000
16	Education - customers	7	11,248	618,663
17	Education - staff	4	6,325	347,900
18	Embedded software	84	252,419	21,455,576
19	Energy consumption monitoring	4	6,325	347,900
20	Energy acquisition	5	7,097	390,350
21	Engineering	79	276,699	20,752,447
22	ERP - Corporate	63	252,982	17,708,755
23	Finances (corporate)	84	210,349	11,569,183
24	Finances (divisional)	63	157,739	8,675,663
25	Governance	10	25,000	1,375,000
26	Government certification (if any)	24	35,571	1,956,383
27	Government regulations (if any)	13	20,003	1,100,155
28	Human resources	7	11,248	618,663
29	Insurance	6	8,935	491,421
30	Inventory management	45	67,003	3,685,140
31	Legal department	24	35,571	1,956,383
32	Litigation	32	47,434	2,608,879
33	Long-range planning	7	18,747	1,031,105
34	Maintenance - product	75	112,484	6,186,627
35	Maintenance - buildings	6	8,435	632,634
36	Manufacturing	178	311,199	23,339,917
37	Market research	38	56,376	3,100,659
38	Marketing	27	39,911	2,195,098
39	Measures - customer satisfaction	4	6,325	347,900
40	Measures - financial	24	35,571	1,956,383
41	Measures - market share	8	12,621	694,151

42	Measures - performance	9	14,161	778,850
43	Measures - quality	10	15,000	825,000
44	Measures - ROI and profitability	32	47,434	2,608,879
45	Mergers and acquisitions	24	59,284	3,260,639
46	Office suites	8	29,449	1,619,686
47	Open-source tools - general	67	100,252	5,513,837
48	Order entry	27	39,911	2,195,098
49	Outside services - manufacturing	24	35,571	1,956,383
50	Outside services - legal	27	66,518	3,658,497
51	Outside services - marketing	15	22,444	1,234,394
52	Outside services - sales	17	25,182	1,385,013
53	Outside services - terminations	9	11,141	612,735
54	Outsource management	32	47,434	2,608,879
55	Patents and inventions	19	28,255	1,554,010
56	Payrolls	21	52,837	2,906,047
57	Planning - manufacturing	42	63,254	3,478,996
58	Planning - products	10	15,000	825,000
59	Process management	12	17,828	980,514
60	Product design	56	140,585	7,732,193
61	Product nationalization	13	30,004	1,650,233
62	Product testing	38	56,376	3,100,659
63	Project offices	32	55,340	3,043,692
64	Project management	10	27,500	1,512,500
65	Purchasing	30	44,781	2,462,941
66	Quality control	13	20,003	1,100,155
67	Real estate	8	12,621	694,151
68	Research and development	106	370,739	20,390,634
69	Sales	45	67,003	3,685,140
70	Sales support	15	22,444	1,234,394
71	Security - buildings	21	31,702	1,743,628

72	Security - computing and software	32	110,680	6,087,384
73	Shareholder relationships	8	29,449	1,619,686
74	Shipping/receiving products	27	66,518	3,658,497
75	Software development	79	238,298	13,106,416
76	Standards compliance	13	20,003	1,100,155
77	Stocks and bonds	21	73,972	4,068,466
78	Supply chain management	47	70,973	3,903,498
79	Taxes	42	84,339	4,638,662
80	Travel	10	25,000	1,375,000
81	Unbudgeted costs - cyber attacks	32	86,963	4,782,945
82	Warranty support	7	10,025	551,384
				-
	Portfolio Totals	**2,366**	**5,192,567**	**308,410,789**

It is obvious that sizing a full portfolio with more than 2,300 applications and more than 5,000,000 function points cannot be accomplished by manual function point counting. At an average counting rate of 500 function points per day, counting this portfolio would take 10,385 days. At a cost of $1,500 per day the expense would be $5,192,197.

This fact alone explains why faster and cheaper function point analysis is a critical step leading to interest in function points by chief executive officers (CEO's) and other C level executives.

Topic 12: Industry Studies using Function Point Metrics

One of the high-interest levels of CEO's and other C level executives is in the area of how their companies compare to others in the same business sector, and how their business sectors compare to other business sectors. Function point metrics are the best choice for these industry studies.

Table 12 shows approximate productivity and quality results for 68 U.S. industries using function points as the basis of analysis:

Table 12: Approximate Industry Productivity and Quality Using Function Point Metrics

	Industry	Software Productivity 2013	Defect Potentials 2013	Removal Efficiency 2013	Delivered Defects 2013
1	Government - intelligence	7.20	5.95	99.50%	0.03
2	Manufacturing - medical devices	7.75	5.20	98.50%	0.08
3	Manufacturing - aircraft	7.25	5.75	98.00%	0.12
4	Telecommunications operations	9.75	5.00	97.50%	0.13
5	Manufacturing - electronics	8.25	5.25	97.00%	0.16
6	Manufacturing - telecommunications	9.75	5.50	96.50%	0.19
7	Manufacturing - defense	6.85	6.00	96.25%	0.23
8	Government - military	6.75	6.40	96.00%	0.26
9	Entertainment - films	13.00	4.00	96.00%	0.16
10	Manufacturing - pharmaceuticals	8.90	4.55	95.50%	0.20
11	Smartphone/tablet applications	15.25	3.30	95.00%	0.17
12	Transportation - airlines	8.75	5.00	94.50%	0.28
13	Software (commercial)	15.00	3.50	94.00%	0.21
14	Manufacturing - automotive	7.75	4.90	94.00%	0.29
15	Transportation - bus	8.00	5.10	94.00%	0.31
16	Manufacturing - chemicals	8.00	4.80	94.00%	0.29
17	Banks - investment	11.50	4.60	93.75%	0.29
18	Open source development	13.75	4.40	93.50%	0.29
19	Banks - commercial	11.50	4.50	93.50%	0.29
20	Credit unions	11.20	4.50	93.50%	0.29
21	Professional support - medicine	8.55	4.80	93.50%	0.31
22	Government - police	8.50	5.20	93.50%	0.34
23	Entertainment - television	12.25	4.60	93.00%	0.32
24	Manufacturing - appliances	7.60	4.30	93.00%	0.30
25	Software (outsourcing)	14.00	4.65	92.75%	0.34
26	Manufacturing - nautical	8.00	4.60	92.50%	0.35
27	Process control	9.00	4.90	92.50%	0.37

28	Stock/commodity brokerage	10.00	5.15	92.50%	0.39
29	Professional support - law	8.50	4.75	92.00%	0.38
30	Games - computer	15.75	3.00	91.00%	0.27
31	Social networks	14.90	4.90	91.00%	0.44
32	Insurance - Life	10.00	5.00	91.00%	0.45
33	Insurance - medical	10.50	5.25	91.00%	0.47
34	Public utilities - electricity	7.00	4.80	90.50%	0.46
35	Education - University	8.60	4.50	90.00%	0.45
36	Automotive sales	8.00	4.75	90.00%	0.48
37	Hospitals	8.00	4.80	90.00%	0.48
38	Insurance - property and casualty	9.80	5.00	90.00%	0.50
39	Oil extraction	8.75	5.00	90.00%	0.50
40	Consulting	12.70	4.00	89.00%	0.44
41	Public utilities - water	7.25	4.40	89.00%	0.48
42	Publishing (books/journals)	8.60	4.50	89.00%	0.50
43	Transportation - ship	8.00	4.90	88.00%	0.59
44	Natural gas generation	6.75	5.00	87.50%	0.63
45	Education - secondary	7.60	4.35	87.00%	0.57
46	Construction	7.10	4.70	87.00%	0.61
47	Real estate - commercial	7.25	5.00	87.00%	0.65
48	Agriculture	7.75	5.50	87.00%	0.72
49	Entertainment - music	11.00	4.00	86.50%	0.54
50	Education - primary	7.50	4.30	86.50%	0.58
51	Transportation - truck	8.00	5.00	86.50%	0.68
52	Government - state	6.50	5.65	86.50%	0.76
53	Manufacturing - apparel	7.00	3.00	86.00%	0.42
54	Games - traditional	7.50	4.00	86.00%	0.56
55	Manufacturing - general	8.25	5.20	86.00%	0.73
56	Retail	8.00	5.40	85.50%	0.78
57	Hotels	8.75	4.40	85.00%	0.66
58	Real estate - residential	7.25	4.80	85.00%	0.72
59	Mining - metals	7.00	4.90	85.00%	0.74
60	Automotive repairs	7.50	5.00	85.00%	0.75
61	Wholesale	8.25	5.20	85.00%	0.78
62	Government - federal civilian	6.50	6.00	84.75%	0.92
63	Waste management	7.00	4.60	84.50%	0.71
64	Transportation - trains	8.00	4.70	84.50%	0.73
65	Food - restaurants	7.00	4.80	84.50%	0.74
66	Mining-coal	7.00	5.00	84.50%	0.78
67	Government - county	6.50	5.55	84.50%	0.86
68	Government - municipal	7.00	5.50	84.00%	0.88
	TOTAL/AVERAGES	**8.95**	**4.82**	**90.39%**	**0.46**

When looking at table 12, the industries that built complex physical devices such as computers, airplanes, medical devices, and telephone switching systems have the best quality. This is because the physical devices won't work unless the software works well with almost zero defects.

For productivity, a different set of industries are at the top including computer games, social networks, entertainment, some commercial vendors (who work many hours of unpaid overtime) and small tablet and smartphone applications built by one or two developers.

Topic 13: Global Studies Using Function Point Analysis

In today's world software development is a global business. About 60% of Indian companies and more than 80% of Indian outsource companies use function points in order to attract outsource business, with considerable success. As already mentioned Brazil now requires function points for all government outsource contracts.

Clearly global competition is a topic of critical interest to all C level executives including CEO's, CFO's, CTO's CIO's, CRO's, and all others.

Function point metrics are the best (and only) metric that is effective for very large scale global studies of software productivity and quality. Table 13 shows approximate results for 68 countries.

Table 13: Approximate Global Productivity and Quality in Function Points

		Approximate Software Productivity (FP per Month)	Approximate Defect Potentials in 2013 (Defects per FP)	Approximate Defect Removal Efficiency	Approximate Delivered Defects in 2013 (Defects per FP)
1	Japan	9.15	4.50	93.50%	0.29
2	India	11.30	4.90	93.00%	0.34
3	Denmark	9.45	4.80	92.00%	0.38
4	Canada	8.85	4.75	91.75%	0.39
5	South Korea	8.75	4.90	92.00%	0.39
6	Switzerland	9.35	5.00	92.00%	0.40
7	United Kingdom	8.85	4.75	91.50%	0.40
8	Israel	9.10	5.10	92.00%	0.41
9	Sweden	9.25	4.75	91.00%	0.43
10	Norway	9.15	4.75	91.00%	0.43
11	Netherlands	9.30	4.80	91.00%	0.43
12	Hungary	9.00	4.60	90.50%	0.44
13	Ireland	9.20	4.85	90.50%	0.46
14	United States	8.95	4.82	90.15%	0.47
15	Brazil	9.40	4.75	90.00%	0.48
16	France	8.60	4.85	90.00%	0.49
17	Australia	8.88	4.85	90.00%	0.49
18	Austria	8.95	4.75	89.50%	0.50
19	Belgium	9.10	4.70	89.15%	0.51
20	Finland	9.00	4.70	89.00%	0.52
21	Hong Kong	9.50	4.75	89.00%	0.52
22	Mexico	8.65	4.85	88.00%	0.58
23	Germany	8.85	4.95	88.00%	0.59
24	Philippines	10.75	5.00	88.00%	0.60
25	New Zealand.	9.05	4.85	87.50%	0.61
26	Taiwan	9.00	4.90	87.50%	0.61

27	Italy	8.60	4.95	87.50%	0.62
28	Jordan	7.85	5.00	87.50%	0.63
29	Malaysia	8.40	4.65	86.25%	0.64
30	Thailand	7.90	4.95	87.00%	0.64
31	Spain	8.50	4.90	86.50%	0.66
32	Portugal	8.45	4.85	86.20%	0.67
33	Singapore	9.40	4.80	86.00%	0.67
34	Russia	8.65	5.15	86.50%	0.70
35	Argentina	8.30	4.80	85.50%	0.70
36	China	9.15	5.20	86.50%	0.70
37	South Africa	8.35	4.90	85.50%	0.71
38	Iceland	8.70	4.75	85.00%	0.71
39	Poland	8.45	4.80	85.00%	0.72
40	Costa Rica	8.00	4.70	84.50%	0.73
41	Bahrain	7.85	4.75	84.50%	0.74
42	Ukraine	9.10	4.95	85.00%	0.74
43	Turkey	8.60	4.90	84.50%	0.76
44	Viet Nam	8.65	4.90	84.50%	0.76
45	Kuwait	8.80	4.80	84.00%	0.77
46	Colombia	8.00	4.75	83.50%	0.78
47	Peru	8.75	4.90	84.00%	0.78
48	Greece	7.85	4.80	83.50%	0.79
49	Syria	7.60	4.95	84.00%	0.79
50	Tunisia	8.20	4.75	83.00%	0.81
51	Saudi Arabia	8.85	5.05	84.00%	0.81
52	Cuba	7.85	4.75	82.50%	0.83
53	Panama	7.95	4.75	82.50%	0.83
54	Egypt	8.55	4.90	82.75%	0.85
55	Libya	7.80	4.85	82.50%	0.85
56	Lebanon	7.75	4.75	82.00%	0.86
57	Iran	7.25	5.25	83.50%	0.87
58	Venezuela	7.50	4.70	81.50%	0.87
59	Iraq	7.95	5.05	82.50%	0.88
60	Pakistan	7.40	5.05	82.00%	0.91
61	Algeria	8.10	4.85	81.00%	0.92
62	Indonesia	8.90	4.90	80.50%	0.96
63	North Korea	7.65	5.10	81.00%	0.97
64	Nigeria	7.00	4.75	78.00%	1.05
65	Bangladesh	7.50	4.75	77.00%	1.09
66	Burma	7.40	4.80	77.00%	1.10
	AVERAGE/TOTAL	**8.59**	**4.85**	**86.27%**	**0.67**

Some of the data in table 13 is provisional and included primarily to encourage more studies of productivity and quality in countries that lack effective benchmarks circa 2013. For example China and Russia are major producers of software but seem to lag India, Brazil, the Netherlands, Finland, and the United States in adopting modern metrics and function points.

Topic 14: Function Points versus Lines of Code (LOC) for Software Economic Analysis

Normally CEO's and other C level executives don't care and may not even know about which programming languages are used for software. However in 1970 within IBM a crisis attracted not only the attention of IBM chairman Thomas J. Watson Jr. but also many other C level executives and vice presidents such as Bob Evans, Vin Learson, Ted Climis, and a number of others.

The crisis was due to the fact that more than half of the schedule and cost estimates in the Systems Development Division and other software groups were wrong and always wrong in the direction of excessive optimism. Worse, the estimates with the largest errors were for projects using the newest and best programming languages such as APL, PL/S, and several others. Only estimates for projects coded in assembly language were accurate.

This crisis was eventually solved and the solution created two useful new concepts: 1) The development of function point metrics by A.J. Albrecht and his colleagues at the IBM White Plains lab; 2) The development of IBM's first parametric estimation tool by the author and Dr. Charles Turk at the IBM San Jose lab. Function points were created in order to provide a language-independent metric for software economic analysis. The IBM development planning system (DPS) was created to estimate projects in any programming language or combination of languages.

For a number of years IBM had used ad hoc estimation ratios based on code development for predicting various non-coding tasks such as design, documentation, integration, testing and the like. When modern languages such as PL/S began to supplant assembly language, it was quickly discovered that ratios no longer worked.

For example the coding effort for PL/S was only half the coding effort for assembly language. The ratio used to predict user documentation had been 10% of coding effort. But for PL/S projects coding effort was cut in two and the manuals were as big as ever, so they were all 50% over budget.

In order to restore some kind of order and rationality to estimates IBM assigned a numeric level to every programming language. Basic assembly was "level 1" and other languages were assigned values based on how many assembly statements would be needed to be equivalent to 1 statement in the target language. Thus Fortran was a level 3 language because it took 3 assembly statements to provide the functionality of 1 Fortran statement. This method used ratios from basic assembly for predicting non coding work.

For several years after the discovery of LOC problems IBM used LOC for coding in the true language for coding the application, but used basic assembly language to derive ratios for non-coding work. This was an awkward method and explains why IBM invested several million

dollars in developing both function point metrics and a formal parametric estimation tool that could handle estimates for applications in all programming languages.

IBM executives were not happy about having to use assembly to show the value of modern languages so they commissioned Al Albrecht and his colleagues to start work on a metric that would be language independent. As we all know function points were the final result.

Table 14.1 shows the numeric levels for a variety of common programming languages. The table also shows the approximate number of logical source code statements per function point:

Table 14.1: Programming Language "Levels" from IBM

Language Levels	Languages	Logical code statements per function point
0.50	Machine language	640.00
1.00	Basic Assembly	320.00
1.45	JCL	220.69
1.50	Macro Assembly	213.33
2.00	HTML	160.00
2.50	C	128.00
3.00	Algol	106.67
3.00	Bliss	106.67
3.00	Chill	106.67
3.00	COBOL	106.67
3.00	Coral	106.67
3.00	Fortran	106.67
3.00	Jovial	106.67
3.25	GW Basic	98.46
3.50	Pascal	91.43
3.50	PL/S	91.43
4.00	ABAP	80.00
4.00	Modula	80.00
4.00	PL/I	80.00
4.50	ESPL/I	71.11
4.50	Javascript	71.11
5.00	Forth	64.00
5.00	Lisp	64.00
5.00	Prolog	64.00
5.00	Basic (interpreted)	64.00
5.25	Quick Basic	60.95
6.00	C++	53.33

6.00	Java	53.33
6.00	PHP	53.33
6.00	Python	53.33
6.25	C#	51.20
6.50	Ada 95	49.23
6.75	RPG III	47.41
7.00	CICS	45.71
7.00	DTABL	45.71
7.00	Ruby	45.71
7.00	Simula	45.71
8.00	DB2	40.00
8.00	Oracle	40.00
8.50	Mixed Languages	37.65
8.50	Haskell	37.65
9.00	Pearl	35.56
9.00	Speakeasy	35.56
10.00	APL	32.00
11.00	Delphi	29.09
12.00	Objective C	26.67
12.00	Visual Basic	26.67
13.00	ASP NET	24.62
14.00	Eiffel	22.86
15.00	Smalltalk	21.33
16.00	IBM ADF	20.00
17.00	MUMPS	18.82
18.00	Forte	17.78
19.00	APS	16.84
20.00	TELON	16.00
25.00	QBE	12.80
25.00	SQL	12.80
50.00	Excel	6.40

For some of the older languages such as COBOL and FORTRAN, the ratios of source code statements to function points were derived by Al Albrecht personally, and then passed into wider usage outside of IBM.

The values for logical code statements per function point only reflect average and the ranges due to individual programming styles can vary by more than 2 to 1 in both directions. This is why mathematical conversion between logical code and function points is not very accurate, in spite of being very easy.

At this point we will discuss a hypothetical software application of 1000 function points in size. We will also make two simplifying assumptions to illustrate the value of function point metrics and the economic problems of lines of code metrics: 1) Coding speed will be assumed to be a constant value of 1000 lines of code per staff month for every language; 2) The total effort for non-coding work such as requirements, design, documentation, management, etc. will be assumed to be an even 50 staffs months of effort. Table 14.2 shows the total effort for building the hypothetical application of 1000 function points using these two assumptions:

Table 14.2: Development Effort for 1000 Function Points
(Assumes a constant rate of 1000 lines of code per month)
(Assumes a constant value of 50 months for non-code work)

Languages	Coding Months	Non-code Months	Total Months
Machine language	640	50	690
Basic Assembly	320	50	370
JCL	221	50	271
Macro Assembly	213	50	263
HTML	160	50	210
C	128	50	178
Algol	107	50	157
Bliss	107	50	157
Chill	107	50	157
COBOL	107	50	157
Coral	107	50	157
Fortran	107	50	157
Jovial	107	50	157
GW Basic	98	50	148
Pascal	91	50	141
PL/S	91	50	141
ABAP	80	50	130
Modula	80	50	130
PL/I	80	50	130
ESPL/I	71	50	121
Javascript	71	50	121
Forth	64	50	114
Lisp	64	50	114
Prolog	64	50	114
Basic (interpreted)	64	50	114
Quick Basic	61	50	111
C++	53	50	103

Java	53	50	103
PHP	53	50	103
Python	53	50	103
C#	51	50	101
Ada 95	49	50	99
RPG III	47	50	97
CICS	46	50	96
DTABL	46	50	96
Ruby	46	50	96
Simula	46	50	96
DB2	40	50	90
Oracle	40	50	90
Mixed Languages	38	50	88
Haskell	38	50	88
Pearl	36	50	86
Speakeasy	36	50	86
APL	32	50	82
Delphi	29	50	79
Objective C	27	50	77
Visual Basic	27	50	77
ASP NET	25	50	75
Eiffel	23	50	73
Smalltalk	21	50	71
IBM ADF	20	50	70
MUMPS	19	50	69
Forte	18	50	68
APS	17	50	67
TELON	16	50	66
QBE	13	50	63
SQL	13	50	63
Excel	6	50	56

Now that we know the total effort for the application of 1000 function points in size, how do we measure economic productivity? The standard definition for economic productivity is "*goods or services produced per unit of labor or expense.*" Let us consider the results for the sum of the coding and non-coding effort using both function points per staff month and lines of code per staff month. Table 12.3 shows both values:

Table 14.3: Function Points versus LOC per Month for Calculating Economic Productivity Rates

Languages	Function Pts. per Month	LOC per Month
Machine language	1.45	927.54
Basic Assembly	2.70	864.86
JCL	3.69	815.29
Macro Assembly	3.80	810.13
HTML	4.76	761.90
C	5.62	719.10
Algol	6.38	680.85
Bliss	6.38	680.85
Chill	6.38	680.85
COBOL	6.38	680.85
Coral	6.38	680.85
Fortran	6.38	680.85
Jovial	6.38	680.85
GW Basic	6.74	663.21
Pascal	7.07	646.46
PL/S	7.07	646.46
ABAP	7.69	615.38
Modula	7.69	615.38
PL/I	7.69	615.38
ESPL/I	8.26	587.16
Javascript	8.26	587.16
Forth	8.77	561.40
Lisp	8.77	561.40
Prolog	8.77	561.40
Basic (interpreted)	8.77	561.40
Quick Basic	9.01	549.36
C++	9.68	516.13
Java	9.68	516.13
PHP	9.68	516.13
Python	9.68	516.13
C#	9.88	505.93
Ada 95	10.08	496.12
RPG III	10.27	486.69
CICS	10.45	477.61
DTABL	10.45	477.61
Ruby	10.45	477.61
Simula	10.45	477.61

DB2	11.11	444.44
Oracle	11.11	444.44
Mixed Languages	11.41	429.53
Haskell	11.41	429.53
Pearl	11.69	415.58
Speakeasy	11.69	415.58
APL	12.20	390.24
Delphi	12.64	367.82
Objective C	13.04	347.83
Visual Basic	13.04	347.83
ASP NET	13.40	329.90
Eiffel	13.73	313.73
Smalltalk	14.02	299.07
IBM ADF	14.29	285.71
MUMPS	14.53	273.50
Forte	14.75	262.30
APS	14.96	251.97
TELON	15.15	242.42
QBE	15.92	203.82
SQL	15.92	203.82
Excel	17.73	113.48

As can easily be seen, the LOC data does not match the assumptions of standard economics, and indeed moves in the opposite direction from real economic productivity. It has been known for many hundreds of years that when manufacturing costs have a high proportion of fixed costs and there is a reduction in the number of units produced, the cost per unit will go up.

The same logic is true for software. When a "Line of Code" is defined as the unit of production, and there is a migration from low-level procedural languages to high-level and object-oriented languages, the number of "units" that must be constructed declines.

The costs of paper documents such as requirements and user manuals do not decline, and tend to act like fixed costs. This inevitably leads to an increase in the "Cost per LOC" for high-level languages, and a reduction in "LOC per staff month" when the paper-related activities are included in the measurements.

On the other hand, the function point metric is a synthetic metric totally divorced from the amount of code needed by the application. Therefore Function Point metrics can be used for economic studies involving multiple programming languages and object-oriented programming languages without bias or distorted results. The Function Point metric can also be applied to non-coding activities such as requirements, design, user documentation, integration, testing, and even project management.

When using the standard economic definition of productivity, which is "*goods or services produced per unit of labor or expense*" it can be seen that the function point ranking matches economic productivity assumptions.

The function point ranking matches economic assumptions because the versions with the lowest amounts of both effort and costs have the highest function point productivity rates and the lowest costs per function point rates.

The LOC rankings, on the other hand, are the exact reversal of real economic productivity rates. This is the key reason why usage of the LOC metric is viewed as **"professional malpractice"** when it is used for cross-language productivity or quality comparisons involving both high-level and low-level programming languages.

The phrase **"professional malpractice"** implies that a trained knowledge worker did something that was hazardous and unsafe, and that the level of training and prudence required to join the profession should have been enough to avoid the unsafe practice.

Since it is obvious that the "lines of code" metric does not move in the same direction as economic productivity, and indeed moves in the opposite direction, it is a reasonable assertion that misuse of lines of code metrics for cross-language comparisons should be viewed as professional malpractice if a report or published data caused some damage or harm.

One of the severe problems of the software industry has been the inability to perform economic analysis of the impact of various tools, methods, or programming languages. It can be stated that the "lines of code" or LOC metric has been a significant barrier that has slowed down the evolution of software engineering, since it has blinded researchers and prevented proper exploration of software engineering factors.

Function point metrics, on the other hand, have opened up many new forms of economic study that were impossible using distorted and inaccurate metrics such as "lines of code" and "cost per defect."

The ability of function point metrics to examine programming languages, methodologies, and other critical software topics is a long step in the right direction.

Topic 15: Function Points and Software Usage and Consumption

One of the newer uses of function point metrics is that of studying software usage and consumption as well as studying software development and maintenance. This field is so new that it has almost no literature except for a paper published by the author of this paper.

It is interesting to start this topic with an overview of how much software an ordinary U.S. citizen uses and owns on a daily basis. Table 15.1 shows approximate software ownership for a fairly affluent person holding down a managerial or technical job:

Table 15.1: U.S. Personal Ownership of Software Circa 2013

Products	Function Points	Hours Used per Day
Home computer	1,000,000	2.50
Tablet	800,000	3.00
Automobile	350,000	3.00
Smart phone	35,000	2.00
Televisions	35,000	4.00
Social networks	20,000	2.50
Medical devices	12,000	24.00
Audio equipment	10,000	1.50
Electronic books	7,500	1.50
Home alarm system	5,000	24.00
Digital camera	3,500	1.00
Hearing aids	3,000	12.00
Digital watches	2,500	12.00
Sum	**2,283,500**	

Only about 50 years ago amount of software owned by anyone would have been close to zero. Today we use software every waking moment, and quite a few devices such a home alarm systems keep working for us while we are asleep, as do embedded medical devices.

The next set of topics where function points are adding insights are the amount of software used by the software engineering and management communities. Table 15.2 shows the approximate amount of software used by software project managers:

Table 15.2: Numbers and Size Ranges of Software Project Management Tools

(Tool sizes are expressed in terms of IFPUG function points, version 4.2)

	Project Management Tools	Lagging	Average	Leading
1	Project planning	1,000	1,250	3,000
2	Project cost estimating			3,000
3	Statistical analysis			3,000
4	Methodology management		750	3,000
5	Reusable feature analysis			2,000
6	Quality estimation			2,000
7	Assessment support		500	2,000
8	Project office support		500	2,000
9	Project measurement			1,750
10	Portfolio analysis			1,500
11	Risk analysis			1,500
12	Resource tracking	300	750	1,500
13	Governance tools			1,500
14	Value analysis		350	1,250
15	Cost variance reporting	500	500	1,000
16	Personnel support	500	500	750
17	Milestone tracking		250	750
18	Budget support		250	750
19	Function point analysis		250	750

20	Backfiring: LOC to FP			300
21	Earned value analysis		250	300
22	Benchmark data collection			300
	Subtotal	***1,800***	***4,600***	***30,000***
	Tools	***4***	***12***	***22***

Project managers in leading or sophisticated companies such as IBM, Google, Microsoft, and the like deploy and use more than 30,000 function points of project management tools.

The next topic of interest shows the volumes of software tools utilized by software engineers themselves. Because software engineering has been highly automated for many years, there is not as large a difference in table 15.3 as there was in table 15.2:

Table15.3: Numbers and Size Ranges of Software Engineering Tools

(Tool sizes are expressed in terms of IFPUG function points, version 4.2)

	Software Engineering Tools	**Lagging**	**Average**	**Leading**
1	Compilers	3,500	3,500	3,500
2	Program generators		3,500	3,500
3	Design tools	1,000	1,500	3,000
4	Code editors	2,500	2,500	2,500
5	GUI design tools	1,500	1,500	2,500
6	Assemblers	2,000	2,000	2,000
7	Configuration control	750	1,000	2,000
8	Source code control	750	1,000	1,500
9	Static analysis (code)		1,500	3,000
10	Automated testing		1,000	1,500
11	Data modeling	750	1,000	1,500
12	Debugging tools	500	750	1,250

13	Data base design	750	750	1,250
14	Capture/playback	500	500	750
15	Library browsers	500	500	750
16	Reusable code analysis			750
	Subtotal	***15,000***	***22,500***	***31,250***
	Tools	***12***	***14***	***16***

If we went through the entire suites of tools used for development, maintenance, testing, quality assurance, technical manuals, and administration we would find that leading software development organizations use about 90 tools that total to more than about 150,000 function points.

Lagging companies use only about 30 tools at a little more than 25,000 function points.

Average companies use about 50 different tools with a total size of perhaps 50,000 function points.

These differences in tool usage patterns also correlate with software quality and productivity levels. However other factors such as team experience, methodologies, and CMMI levels are also correlated with higher productivity and quality so it is not yet possible to isolate just the impacts of tools themselves.

From analysis of more than 15,000 software projects various patterns have been noted of tools and methods that are used by successful projects. The definition of "success" includes productivity and quality results > 25% better than average for the same size and class of software, combined with schedules about 15% shorter than average. These patterns of success include:

Patterns of Tools Noted on Successful Software Projects

1. TSP, RUP, or hybrid as the development methods for large applications
2. Agile, XP, iterative, Prince2 or defined methods for small applications
3. Achieving > CMMI 3 for defense projects
4. Early sizing of projects using automated tools
5. Early risk analysis of projects before starting
6. Early quality predictions using automated tools
7. Early use of parametric estimation tools for cost and schedule predictions

8. Use of automated project management tools for team assignments
9. Use of automated project office support tools for large projects
10. Use of automated requirements modeling tools for critical projects
11. Use of certified collections of reusable materials (design, code, test cases, etc.)
12. Use of static analysis tools for code in all languages supported by static analysis
13. Use of inspections and inspection support tools for requirements and design
14. Use of certified test personnel for critical applications
15. Use of cyclomatic complexity code analysis tools
16. Use of test coverage tools that show requirements and path coverage
17. Use of automated test tools for unit, function, regression, and other tests
18. Use of formal mathematical test case design methods
19. Use of formal change control and change control boards
20. Accurate status reports that highlight potential problems

Software projects at the leading edge usually range from about 12 to more than 25 function points per staff month in terms of productivity.

Defect potentials on leading projects are < 3.00 per function point combined with defect removal efficiency levels that average > 97% for all projects and 99% for mission-critical projects.

Excellence in quality control and change control leads to more than 95% of these projects being finished and delivered. Those that are not delivered are terminated for business reasons such as mergers, acquisitions, or divestitures. More than 90% of projects in this class are on time and within planned budgets, and about 15% are slightly faster and below planned budgets.

When you consider the other end of the spectrum or software projects that have worse than average results, this is what you find. We are now dealing with the opposite side of a bell-shaped curve, and considering projects that are < 25% worse than average for the same size and class of software, combined with schedules about 15% longer than average. The patterns of failing projects include:

Patterns of Tools Noted on Unsuccessful Projects

1. Waterfall development methods for large applications
2. Cowboy or undefined methods for small applications
3. CMMI 1 for defense projects
4. No early sizing of projects using automated tools
5. No early risk analysis of projects before starting
6. No early quality predictions using automated tools
7. Manual and optimistic estimation methods for cost and schedule predictions
8. No use of automated project management tools for team assignments
9. No use of automated project office support tools for large projects
10. No use of automated requirements modeling tools for critical projects
11. No use of certified collections of reusable materials (design, code, test cases, etc.)

12. No use of static analysis tools for code in any language
13. No use of inspections and inspection support tools for requirements and design
14. No use of certified test personnel for critical applications
15. No use of cyclomatic complexity code analysis tools
16. No use of test coverage tools that show requirements and path coverage
17. Little use of automated test tools for unit, function, regression, and other tests
18. No use of formal mathematical test case design methods
19. No use of formal change control and change control boards
20. Inaccurate status reports that conceal potential problems

Software projects at the trailing edge usually range from about 3 to perhaps10 function points per staff month in terms of productivity.

Poor quality is the main reason for schedule slips and cost overruns. Defect potentials are usually > 5.00 per function point combined with defect removal efficiency levels that almost always average < 85% for all projects and seldom top 90% even for mission-critical projects. In some cases defect removal efficiency on lagging projects drops below 80%. These dismal projects do not use either inspections or static analysis and terminate testing prematurely due to not understanding quality economics.

Exploration of software consumption and software tool usage should be a valuable new form of research for the software engineering and function point communities.

Topic 16: Function Points and Software Outsource Contracts

The government of Brazil already requires function point metrics for all contracts involving software. Both South Korea and Italy may soon do the same. Function point metrics are an excellent choice for software outsource agreements.

Some of the topics where function points should be used in contracts would include, but are not limited to:

- Cost per function point for fixed-price contracts.
- Work hours per function point for time and materials contracts.
- Delivered defect densities expressed in terms of defects per function point.
- Function points combined with defect removal efficiency (DRE). Contracts should require that the vendor top a specific level of DRE such as 97%, measured by counting internal defects and comparing them to user-reported defects in the first 90 days of usage.

Software Risk Master (SRM) includes estimates for both the odds of litigation occurring and also for the probable legal expenses for both the plaintiff and the defendant. For example if an outsource contract for an application of 10,000 function points goes to court for breach of contract, the probable costs to the plaintiff will be about $7,500,000 and the probable costs to the defendant will be about $9,000,000 if the case goes through trial. (About 90% settle out of court.) Of course whichever side loses will have much higher costs due to probable damages and perhaps paying court costs for both sides.

Software Risk Master predicts attorney fees, paralegal fees, and expert witness fees. It also predicts the lost time for executives and technical staff when they are involved with discovery and depositions. SRM also predicts the probable month the litigation will be filed, and the probable duration of the trial.

About 90% of lawsuits settle out of court. SRM cannot predict out of court settlements since many of these are sealed and data is not available.

In the modern era an increasingly large number of organizations are moving toward outsourcing or the use of contractors for development or maintenance (or both) of their software applications. Although the general performance of outsourcing vendors and contract software development organizations is better than the performance of the clients they serve, it is not perfect.

When software is developed internally within a company and it runs late or exceeds its budget, there are often significant disputes between the development organization and the clients who commissioned the project and are funding it, as well as the top corporate executives. Although these internal disputes are unpleasant and divisive, they generally do not end up in court under litigation.

When software is developed by a contractor and runs late or exceeds the budget, or when it is delivered in less than perfect condition, the disputes have a very high probability of moving to litigation for breach of contract. From time to time, lawsuits may go beyond breach of contract and reach the point where clients charge fraud.

As international outsourcing becomes more common, some of these disputes involve organizations in different countries. When international laws are involved, the resolution of the disputes can be very expensive and protracted. For example some contracts require that litigation be filed and use the laws of other countries such as Hong Kong or China.

The author has often commissioned to perform independent assessments of software projects where there is an anticipation of some kind of delay, overrun, or quality problem. He has also been engaged to serve as expert witnesses in a dozen lawsuits involving breach of contract between clients and software contractors. He has also been engaged to work as an expert in software tax cases.

From participating in a number of such assessments and lawsuits, it is obvious that most cases are remarkably similar. The clients charge that the contractor breached the agreement by delivering the software late, by not delivering it at all, or by delivering the software in inoperable condition or with excessive errors.

The contractors, in turn, charge that the clients unilaterally changed the terms of the agreement by expanding the scope of the project far beyond the intent of the original agreement. The contractors also charge some kind of non-performance by the clients, such as failure to define requirements or failure to review delivered material in a timely manner.

The fundamental root causes of the disagreements between clients and contractors can be traced to two problems:

- Ambiguity and misunderstandings in the contract itself.

- The historical failure of the software industry to quantify the dimensions of software projects before beginning them.

Although litigation potentials vary from client to client and contractor to contractor, the overall results of outsourcing within the United States approximates the following distribution of results after about 24 months of operations, as derived from observations among the author's clients:

Table 16.1: Approximate Distribution of U.S. Outsource Results after 24 Months

Results	Percent of Outsource Arrangements
Both parties generally satisfied	70%
Some dissatisfaction by client or vendor	15%
Dissolution of agreement planned	10%
Litigation between client and contractor probable	**4%**
Litigation between client and contractor in progress	**1%**

Table 16.1 shows all projects and all contracts. The odds of litigation rise steeply with application size:

Table 16.2: Odds of Outsource Litigation by Application Size

10 function points	< 0.5% chance of litigation
100 function points	< 1% chance of litigation
1000 function points	< 3% chance of litigation
10,000 function points	**+ or – 12% chance of litigation**
100,000 function points	**+ or – 25% chance of litigation**

As if 2013 the software industry does not really know how to build large software projects well. Far too many are terminated, don't work when delivered, or end up in court for breach of contract. The actual technologies for building large systems exist, but less than5% of companies know them based on on-site discussions with executives and development teams. This is true of outsource companies as well as commercial developers, in-house IT systems, and even games.

From process assessments performed within several large outsource companies, and analysis of projects produced by outsource vendors, our data indicates better than average quality control approaches when compared to the companies and industries who engaged the outsource vendors.

Software estimating, contracting, and assessment methodologies have advanced enough so that the root causes of software outsource contracts can now be overcome. Software estimation is now sophisticated enough so that a formal estimate using one or more of the commercial

parametric software estimation tools in conjunction with software project management tools can minimize or eliminate unpleasant surprises later due to schedule slippages or cost overruns.

Indeed, old-fashioned purely manual cost and schedule estimates for major software contracts should probably be considered an example of professional malpractice. Manual estimates are certainly inadequate for software contracts or outsource agreements whose value is larger than about $500,000.

A new form of software contract based on the use of function point metrics is clarifying the initial agreement and putting the agreement in quantitative, unambiguous terms. This new form of contract can also deal with the impact of creeping user requirements in a way that is agreeable to both parties. As mentioned earlier the government of Brazil now requires function point metrics for all software contracts.

For major software contracts involving large systems in excess of 10,000 function points independent assessments of progress at key points may also be useful.

As stated, the author has been an expert witness in a dozen breach of contract lawsuits. The four most common reasons for breach of contract include:

1. Optimistic estimates by the vendor before starting.
2. Inadequate quality control and bypassing inspections and static analysis.
3. Inadequate change control in the face of > 2% requirements creep per month.
4. Project managers concealing problems from both clients and their own executives.

Outsource contracts are often poorly formed and contain clauses that don't make sense. For example a contract between a vendor and a State government included a clause that required the vendor to deliver "zero defect" software. This was technically impossible and should not have been in the contract. The vendor should never have agreed to this, and the vendor's lawyer was flirting with professional malpractice to allow such a clause to remain.

Other cases where function points have been useful in deciding the issues include the following:

A Canadian case involved 82 major changes which doubled the size of an application from 10,000 to 20,000 function points. The client refused to pay claiming that the changes were "elaborations" and not new features. The court decided that since function points measure features, the 82 changes were in fact new features and ordered the defendant to pay the vendor.

An arbitration in Hong Kong involved adding 13,000 function points to an application late in development. The contract was a fixed-price contract. Since it is a proven fact that late changes cost more than original work, the vendor was asking for additional fees to recover the higher costs. Following is the author's suggested format for a project status report.

Note that the first topic each month will be a discussion of "red flag" items that might throw off the schedule or costs of the project:

Suggested Format for Monthly Status Reports for Software Projects

1. **Status of last months "red flag" problems**
2. **New "red flag" problems noted this month**
3. Change requests processed this month versus change requests predicted
4. Change requests predicted for next month
5. Size in function points for this month's change requests
6. Size in function points predicted for next month's change requests
7. Schedule impacts of this month's change requests
8. Cost impacts of this month's change requests
9. Quality impacts of this month's change requests
10. Defects found this month versus defects predicted
11. Defects predicted for next month
12. Costs expended this month versus costs predicted
13. Costs predicted for next month
14. Deliverables completed this month versus deliverables predicted
15. Deliverables predicted for next month

Although the suggested format somewhat resembles the items calculated using the earned value method, this format deals explicitly with the impact of change requests and also uses function point metrics for expressing costs and quality data.

An interesting question is the frequency with which milestone progress should be reported. The most common reporting frequency is monthly, although exception reports can be filed at any time that it is suspected that something has occurred that can cause perturbations. For example, serious illness of key project personnel or resignation of key personnel might very well affect project milestone completions and this kind of situation cannot be anticipated.

It might be thought that monthly reports are too far apart for small projects that only last six months or less in total. For small projects weekly reports might be preferred. However, small projects usually do not get into serious trouble with cost and schedule overruns, whereas large projects almost always get in trouble with cost and schedule overruns. This article concentrates on the issues associated with large projects. In the litigation where the author has been an expert witness, every project under litigation except one was larger than 10,000 function points in size.

Daily scrum sessions, while useful and interesting, have no legal standing in the case of litigation. Due to the absence of minutes or notes of what transpired, scrum sessions can make litigation complex.

For outsource projects under contract a formal status report signed by project managers would be the best technical choice. The author is not an attorney and this should not be construed as legal advice. For outsource contracts seek the advice of an attorney on the need for formal written

status reports. Table 16.3 shows potential costs for the plaintiff for an application of 10,000 function points in size that ends up in breach of contract litigation:

Table 16.3: Plaintiff Outsource Litigation Analysis

Project function points	10,000
Consequential damages	**$25,118,864**
Attorney hourly $	$400.00
Paralegal hourly $	$150.00
Expert hourly $	$450.00
Executive hourly $	$200.00
Staff hourly $	$75.00
Planned project duration	34
Probable month of filing	40
Probable trial duration	24
Odds of out of court settlement	**83.00%**
Plaintiff legal fees	$4,000,000
Plaintiff paralegal fees	$1,500,000
Plaintiff expert fees	$337,500
Plaintiff executive costs	$1,500,000
Plaintiff staff costs	$412,500
Total	$7,750,000
Consequential damages per FP	$2,511.89
Litigation $ per FP	$775.00

Table 16.4 shows the same 10,000 function point application, but the probable costs for the defendant are shown instead of the plaintiff:

Table 16.2: Defendant Litigation Analysis

Project function points	10,000
Damages if suit is lost	**$57,987,729**
Attorney hourly $	$400.00
Paralegal hourly $	$150.00
Expert hourly $	$450.00

Executive hourly $	$200.00
Staff hourly $	$75.00
Planned project duration	34
Probable month of filing	40
Probable trial duration	24
Odds of out of court settlement	**83.00%**
Plaintiff legal fees	$4,800,000
Plaintiff paralegal fees	$1,650,000
Plaintiff expert fees	$495,000
Plaintiff executive costs	$1,700,000
Plaintiff staff costs	$487,500
Total	$9,132,500
Potential $ if suit is lost	**$74,870,228**
Litigation $ per FP	$913.25
$ per FP if suit is lost	$7,487

The information in tables 16.1 through 16.4 is generic and not based on any specific case. Defendant costs are often higher than those of the plaintiff because the downside of losing a major breach of contract lawsuit can be very expensive indeed.

Once again, the author is not an attorney and is not providing any legal advice. Readers considering litigation for software projects in trouble should seek legal advice from attorneys. But having worked as an expert witness in a number of breach of contract cases, it seems much better to have a good contract before starting and to have a successful project outcome. Litigation is costly and time consuming for both parties. The technologies of software are good enough so that almost all projects could be successful if the vendors actually know and use state of the art methods.

For example a way of minimizing the odds of litigation would be to include a clause in outsource contracts that require the outsource vendor to record development defects and a mandate that defect removal efficiency (DRE) should top 97% measured by comparing internal defects and customer-reported defects; i.e. the vendor should guarantee that at least 97 out of every 100 bugs were removed. This is technically possible, and high levels of DRE speed schedules and lower both development and maintenance costs.

Topic 17: Function Points and Venture Funding of Software Startups

Every year there are hundreds of new software companies starting up. Many of these receive funding from venture capitalists. Some of these companies grow to become hugely successful such as Microsoft, Google, and Facebook. However more than 90% of these software startup companies fail.

A common reason for failure is that the startup companies burn through several rounds of financing because they are late in developing and bringing their product to market. Software Risk Master (SRM) has a standard feature that predicts the development costs and schedules for new software applications. If these applications are venture funded, SRM also has a standard feature for predicting the number of rounds of venture financing needed, as well as the equity dilution for the entrepreneurs.

As an example, the State of Rhode Island acted as a venture capitalist for Curt Schilling's game company, Studio 38. The state did not perform due diligence nor did it realize that more than one round of funding would be needed.

In the aftermath of the Studio 38 bankruptcy SRM was used to carry out a retroactive post mortem. SRM predicted an 88% chance of failure for Studio 38. It also predicted that the total amount of funding needed would not just be $75,000,000 for the first release, but instead $206,000,000 would be needed when post-release maintenance and quality control costs were included. These risk and cost predictions took only seven minutes.

Tables 17.1 through 17.3 illustrate the risk predictions and venture funding feature for three software applications, all of 1000 function points in size. Table 17.1 shows a best-case scenario with a top-gun team using state of the art methods. Table 17.2 shows an average team and average methods. Table 17.3 shows an unsophisticated team using unsafe methods. Note that 1000 function points is fairly small, but not unusual for the first release of a new commercial software package.

Table 17.1: Risks and Venture Funding for a Top Software Team

		Venture
Risks		**Investment**
Cancellation	**6.73%**	**$574,718**
Negative ROI	**8.52%**	**Rounds**
Cost overrun	**7.63%**	1
Schedule slip	**8.97%**	**Total Equity**
Unhappy clients	**10.32%**	**$919,548**
Litigation	**3.14%**	**Dilution**
Average Risks	**7.55%**	**41.50%**
Financial Risk	**14.05%**	**Ownership**
		58.50%

The investment cost per function point for the best-case scenario is $919.58 which encompasses both software development and also support functions such as marketing, sales, administration, and management.

Table 17.2 shows exactly the same size and type of software application, but developed by an average team:

Table 17.2: Risks and Venture Funding for an Average Software Team

Risks		**Venture Investment**
Cancellation	**11.73%**	**$1,656,443**
Negative ROI	**14.86%**	**Rounds**
Cost overrun	**13.30%**	2
Schedule slip	**15.64%**	**Total Equity**
Unhappy clients	**17.99%**	**$2,815,954**
Litigation	**5.48%**	**Dilution**
Average Risks	**13.17%**	**55.50%**
Financial Risk	**24.50%**	**Ownership**
		44.50%

The investment cost per function point for the average case is $2,815.95. This is the investment needed for the total company including software, marketing, sales, administration, and management. In other words, function points can be applied to software corporate startups if software is the main and only product. The CEO's in this case would be the actual entrepreneurs.

Table 17.3 shows the risks and venture funding for the same size and type of software project, but with an inexperienced team using marginal methods:

Table 17.3: Risks and Venture Funding for a Marginal Software Team

Risks		**Venture Investment**
Cancellation	**17.84%**	**$6,206,913**
Negative ROI	**22.60%**	**Rounds**
Cost overrun	**20.22%**	3
Schedule slip	**23.79%**	**Total Equity**
Unhappy clients	**27.36%**	**$10,551,752**
Litigation	**8.33%**	**Dilution**
Average Risks	**20.02%**	**78.00%**
Financial Risk	**37.25%**	**Ownership**
		22.00%

The investment for the worst-case scenario is a shocking $10,551.72. It is unlikely that professional venture capitalists would put so much money into such a poorly staffed and organized group. Almost certainly the funds would stop before the third round and the company would go bankrupt without making their initial delivery.

The early failure of venture-backed software companies is harmful to both the entrepreneurs and to the venture capitalists. Software Risk Master (SRM) can be used prior to any actual investment and show both parties the probable schedules, costs, team size, and quality to bring the first release to market. SRM can also predict post-release enhancements and maintenance of five years. It also predicts numbers of bugs that might be released, and the customer support costs needed to deal with them.

Topic 18: Function Points for Analysis of Software Occupation Groups

Several years ago the author was commissioned by AT&T to perform a study on the various kinds of occupation groups employed by large organizations that produced software. Some of the participants in the study included AT&T itself, IBM, Ford Motors, Texas Instruments, the U.S. Navy, and a dozen other somewhat smaller organizations.

The study found a total of 116 different occupations associated with software. No individual project employs all 116, but several large systems in large companies have employed 50 different occupations. In fact for large systems pure programming may be less than 25% of the total effort.

The study also encountered some interesting sociological phenomena:

- Not a single Human Resource organization in either corporations or government agencies actually knew how many software personnel were employed. It was necessary to interview unit management to find out.

- A surprising number of engineers building embedded software refused to be called "software engineers" and insisted on their academic titles such as electrical engineer, automotive engineer, aeronautical engineer or whatever it was. The reason for this is because "software engineering" does not have the same professional respect among C level executives as the other forms of engineering.

- Due to the fact that some software personnel refused to be identified as software engineers and no HR group knew software employment, we can assume that the Department of Commerce statistics on U.S. software employment are probably wrong, and wrong by undercounting the engineers who refuse to accept software engineering job descriptions.

One of the many problems with the older "lines of code" or LOC metric is that it cannot be used to measure the performance of business analysts, quality assurance, technical writers, project managers or any of the 116 occupation groups other than pure programmers.

Function point metrics, on the other hand, can be used to measure the performance of all 116 occupations. There are two key methods for doing this. The first is to use function points to measure the "assignment scope" or the amount of work assigned to one person in a specific occupation. The second is to use function points to measure the "production rate" or the amount of work one person can perform in a given time period such as a calendar month.

Here is a small example just to illustrate the points. Assume the project under development is 1,000 function points in size.

To measure the work of software quality assurance specialists (SQA) assume that the assignment scope is 500 function points. This means that two SQA personnel will be needed for the total application of 1,000 function points.

Assume that the production rate of the two SQA personnel is each 250 function points per calendar month. That means that each SQA person will need two calendar months to complete their analysis of the quality of their portion of the application.

Putting both sets of measures together, the application of 1,000 function points needed 2 SQA personnel who together worked for a total of 4 months. The net productivity for SQA in this case would be 1,000 function points divided by 4 months of effort or 250 function points per staff month. Using the reciprocal measure of work hours per function point, the SQA effort would be 1.89 work hours per function point. Note that these are merely examples to illustrate the math and should not be used for actual estimates.

The main point is that function point metrics are the only available metric that can analyze the contributions of all 116 different occupation groups. Table 18 lists the 116 occupations in alphabetical order:

Table 19: Software Specialization Circa 2013

1. Accounting/Financial Specialists
2. Agile coaches
3. Architects (Software)
4. Architects (Systems)
5. Architects (Enterprise)
6. Assessment Specialists
7. Audit Specialists
8. Baldrige Award Specialists
9. Baselining Specialists
10. Benchmarking Specialists
11. Business analysts (BA)
12. Business Process Reengineering (BPR) Specialists
13. Capability Maturity Model Integrated (CMMI) Specialists
14. CASE and tool Specialists
15. Client-Server Specialists
16. CMMI Assessors
17. Complexity Specialists
18. Component Development Specialists
19. Configuration Control Specialists
20. Cost Estimating Specialists
21. Consulting Specialists
22. Curriculum Planning Specialists

23. Customer Liaison Specialists
24. Customer Support Specialists
25. Data Base Administration Specialists
26. Data Center Support Specialists
27. Data quality Specialists
28. Data Warehouse Specialists
29. Decision Support Specialists
30. Development specialists
31. Distributed Systems Specialists
32. Domain Specialists
33. Earned Value Specialists
34. Education Specialists
35. E-Learning Specialists
36. Embedded Systems Specialists
37. Enterprise Resource Planning (ERP) Specialists
38. Executive Assistants
39. Frame Specialists
40. Expert-System Specialists
41. Function Point Specialists (certified)
42. Generalists (who perform a variety of software-related tasks)
43. Globalization and Nationalization Specialists
44. Graphics Production Specialists
45. Graphical User Interface (GUI) Specialists
46. Human Factors Specialists
47. Information Engineering (IE) Specialists
48. Instructors (Management Topics)
49. Instructors (Software Topics)
50. Integration Specialists
51. Intellectual Property (IP) Specialists
52. Internet specialists
53. ISO Certification Specialists
54. Joint Application Design (JAD) Specialists
55. Kanban Specialists
56. Kaizen Specialist s
57. Knowledge specialists
58. Key Process Indicators (KPI) specialists
59. Library Specialists (for project libraries)
60. Litigation support Specialists
61. Maintenance Specialists
62. Marketing Specialists
63. Member of the Technical Staff (multiple specialties)
64. Measurement Specialists
65. Metric Specialists
66. Microcode Specialists
67. Model Specialists
68. Multi-Media Specialists
69. Network maintenance Specialists
70. Network Specialists (LAN)

71. Network Specialists (WAN)
72. Network Specialists (Wireless)
73. Neural Net Specialists
74. Object-Oriented Specialists
75. Outsource Evaluation Specialists
76. Package Evaluation Specialists
77. Pattern Specialists
78. Performance Specialists
79. Programming Language Specialists (Java, C#, Ruby, PHP, SQL, etc.)
80. Project Cost Analysis Specialists
81. Project managers
82. Project Office Specialists
83. Project Planning Specialists
84. Process Improvement Specialists
85. Productivity Specialists
86. Quality Assurance Specialists
87. Quality function deployment (QFD) Specialists
88. Quality Measurement Specialists
89. Rapid Application Development (RAD) Specialists
90. Research Fellow Specialists
91. Reliability Specialists
92. Repository Specialists
93. Reengineering Specialists
94. Requirements engineer
95. Reverse engineering Specialists
96. Reusability Specialists
97. Reverse Engineering Specialists
98. Risk Management Specialists
99. Sales Specialists
100. Sales Support Specialists
101. Scrum masters
102. Security Specialists
103. Standards Specialists
104. Systems Analysis Specialists
105. Systems Support Specialists
106. Technical Translation Specialists
107. Technical Writing Specialists
108. Test Case Design Specialists
109. Testing Specialists (Automated)
110. Testing Specialists (Manual)
111. Testing Specialists (Model Driven)
112. Total Quality Management (TQM) Specialists
113. Virtual Reality Specialists
114. Web Development Specialists
115. Web Page Design Specialists
116. Web Masters

Software engineering is following a similar path as did the older forms of engineering, and for that matter of medicine and law. The path includes more and more granular forms of specialization.

As more different kinds of specialists appear and begin to work on large and complex software applications, pure coding is no longer the major activity. In fact many large applications create more English words by far than they do code, and the costs of the words are much higher. Some military software projects have been measured at creating about 400 English words for every Ada statement!

Function point metrics are useful in analyzing the overall performance of the occupation groups employed on large and complex software systems.

As an example of the diversity of occupations on large software applications, table 19.1 shows the pattern of occupation groups for a very large system of 100,000 function points. Table 19.1 is a standard output from Software Risk Master ™ (SRM):

Table 19.1: Occupation Groups and Part-Time Specialists
(Application size = 100,000 function points)

	Normal Staff	Peak Staff
Programmers	290	43
Testers	256	38
Designers	138	22
Business analysts	138	21
Technical writers	60	8
Quality assurance	15	8
1st line managers	54	8
Data base administration	26	6
Project Office staff	32	4
Administrative support	26	3
Configuration control	15	3
Project librarians	21	2
2nd line managers	9	7
Estimating specialists	9	3
Architects	6	2
Security specialists	3	9
Performance specialists	3	5
Function point counters	3	5
Human factors specialists	3	5
3rd line managers	2	5
Total Staff	**1,119**	**1,680**

Although programmers are the largest occupation group with an average size of 290 personnel, that is only 25.92% of the total personnel employed. Large systems use many occupations.

Topic 19: Data Used by Fortune 500 C-Level Executives

Large corporations in the Fortune 500 class spend between about $1,000,000 and $6,000,000 per year for various kinds of benchmarks. Most companies don't actually know their benchmark costs because they are scattered across all operating units. There is no central reporting or consolidation of benchmark cost data. Only consultants who visit a number of business units realize how many different kinds of benchmarks are used in large companies.

- Human resource groups use benchmarks on compensation levels. They also perform internal benchmark studies on morale.
- Legal groups use benchmarks on patent and other forms of litigation.
- Marketing and sales groups use benchmarks on competitive products and market shares. They also carry out proprietary benchmark studies of customer satisfaction.
- Manufacturing groups use benchmarks on cost per unit and manufacturing speed.
- Computer operation groups use data center benchmarks.
- Engineers use many different kinds of hardware benchmarks.
- Purchasing groups use benchmarks on pricing ranges for standard parts and devices.
- Software groups use benchmarks on productivity, quality, maintenance, and other topics.

Table 19 summarizes the interest levels in 70 different kinds of benchmarks by a sample of C-level executives (CEO and CFO) from several Fortune 500 companies studied by the author:

Table 19: Software Benchmarks Used by Fortune 500 C-Level Executives

	Software Benchmarks	CEO Interest	CFO Interest	Best metrics used for benchmarks
1	Competitive practices within industry	10	10	**$ per function point**
2	Project failure rates (size, methods)	10	10	**Function points**
3	Risks: Software	10	10	**Function points + DRE**
4	Patent litigation and results	10	10	Cases won, lost; specific issues
5	Outsource contract success/failure	10	10	Function points, defect removal
6	Risks: corporate/financial	10	10	Dollars/litigation/competition
7	Risks: Legal	10	10	**Function points, defect removal**
8	Cyber Security attacks (number, type)	10	10	Vulnerabilities; defense
9	Return on investment (ROI)	10	10	**Function points, ROI**
10	Total cost of ownership (TCO)	10	10	**Function points, ROI**
11	Customer satisfaction	10	10	Percentage of satisfied clients
12	Data quality	10	10	Data points (hypothetical)
13	Cost of Quality (COQ)/technical debt	10	10	**Function points, defect removal**
14	Development costs: major projects	10	10	**Function points**
15	Litigation - canceled projects/poor quality	10	10	No standard metrics
16	Litigation - intellectual property	10	10	No standard metrics

17	Portfolio size, maintenance costs	10	10	**Function points**
18	Litigation - breach of contract	10	10	No standard metrics
19	Software development Benchmarks	9	10	**Function points**
20	Occupation group compensation	9	10	Average $ per occupation
21	ERP installation/customization	9	10	**Function points/data points**
22	Data center benchmarks	9	10	$ per transaction; transactions speed
23	Occupation groups by industry, size	9	10	Occupations by industry
24	Employee morale	10	9	Percentage of satisfied workers
25	Team compensation level	10	9	Compensation by occupation
26	Attrition by occupation, size, industry	10	9	Attrition % by job title
27	CMMI assessments within organization	9	9	Key process indicators (KPI)
28	Customer support benchmarks	9	9	Customers served per time unit
29	Enhancement costs	8	10	**Function points**
30	Skills inventories by occupation	8	9	Skill list
31	Best Practices - maintenance	8	9	**Function points**
32	Maintenance costs (annual)	8	9	**Function points, defect removal**
33	Data base size	9	8	Data points (hypothetical_
34	Industry productivity	10	7	**Function points**
35	Team morale	9	8	Percentage by occupation group
36	ISO standards certification	9	7	No standard metrics
37	Productivity - project	8	8	**Function points**
38	Technical debt	8	8	**Function points, defect removal**
39	Application sizes by type	8	8	**Function points**
40	Best Practices - requirements	6	9	**Function points**
41	Team attrition rates	7	8	Percentage by job title
42	Best Practices - test efficiency	8	7	Function points, defect removal
43	Coding speed in LOC	7	8	Lines of code (LOC)
44	Litigation - employment contracts	7	8	No standard metrics
45	Code quality (only code - nothing else)	7	7	Lines of code (LOC)
46	Application types	7	7	Taxonomy
47	Cost per defect (caution: unreliable)	6	7	Cost per defect; **cost per FP**
48	Software maintenance/serviceability	6	7	**Function points/complexity**
49	Hardware performance benchmarks	6	7	MIPS
50	Country productivity	8	5	**Function points**
51	Best Practices - pre-test defects	6	6	**Function points, defect removal**
52	Earned value (EVA)	5	7	**Function points**
53	Best Practices - defect prevention	5	6	**Function points, defect prevention**
54	Methodologies: Agile, RUP, TSP, etc.	5	6	**Function points**
55	Methodology comparisons	5	6	**Function points**
56	CMMI levels within industries	6	5	Percentage by CMMI levels
57	Test coverage benchmarks	5	6	Requirements, control flow coverage
58	Best Practices - design	5	5	**Function points**

59	Productivity - activity	4	6	**Function points/activities**
60	Standards benchmarks	5	5	**Function points, defect removal**
61	DCUT benchmarks: function points	3	7	**Function points**
62	Application class by taxonomy	4	5	Taxonomy
63	Serviceability benchmarks	4	5	Maintenance assignment scope
64	Tool suites used	4	4	**Function points by tool types**
65	SNAP non- functional size metrics	3	4	**SNAP plus normal function points**
66	Metrics used by company	3	3	Percentage by metric
67	Programming Languages used	3	3	Language levels
68	Certification benchmarks	3	3	**Function points, defect removal**
69	DCUT benchmarks: LOC	1	2	Logical code statements
70	Cyclomatic complexity benchmarks	1	1	Cyclomatic complexity

This set of corporate benchmark uses many different metrics. However function point metrics are used for 33 benchmarks out of the total of 70 shown. As of 2013 there is no other metric that is more widely used for software than function point metrics. There is no metric that is more reliable or more accurate for software economic analysis than function point metrics. With more than 50,000 software projects measured using function points, the volume of function point benchmark data is larger than all other metrics combined.

- Lines of code metrics are useless for requirements and design analysis and they also penalize high-level languages. They are harmful for economic studies covering multiple programming languages.
- Cost per defect penalizes quality and does not measure the value of quality.
- Story points are not standardized and have no major collections of benchmark data.
- Use-case points are useful for applications that use the UML and use cases, but worthless for other kinds of software.

Function point metrics are the best metric yet developed for understanding software productivity, software quality, and software economics.

Topic 20: Combining Function Points with other Metrics

Although function point metrics are powerful and have many uses, they are not the only useful metric for software projects. This topic illustrates how function points can be combined with other metrics to improve overall understanding of software quality and software economics.

Function Points and Defect Removal Efficiency (DRE)

Software quality is the weak link of software engineering. In general about 50 cents out of every dollar spent on software goes to finding and fixing bugs. Metrics for evaluating software quality such as "cost per defect" have been inaccurate and fail to show true quality economics. Quality metrics such as "defects per KLOC" ignore requirements and design defects, which outnumber code defects for large software systems.

Function point metrics combined with defect removal efficiency (DRE) provide the strongest and most accurate set of quality metrics yet developed. DRE is combined with a function-point value called "defect potential" or the total numbers of bugs that are likely to found. Here too this metric originated in IBM in the early 1970's. In fact the author was on one of the original IBM teams that created this metric and collected data from internal projects.

The defect removal efficiency (DRE) metric was developed in IBM in the early 1970's at the same time IBM was developing formal inspections. In fact this metric was used to prove the efficiency of formal inspections compared to testing by itself.

The concept of DRE is to keep track of all bugs found by the development teams and the compare those bugs to post-release bugs reported by customers in a fixed time period of 90 days after the initial release.

If the development team found 900 bugs prior to release and customer reported 100 bugs in the first three months, then the total volume of bugs was an even 1,000 so defect removal efficiency is 90%. This combination is simple in concept and powerful in impact.

The U.S. average circa 2013 for defect removal efficiency (DRE) is just a bit over 85%. Testing alone is not sufficient to raise DRE much above 90%. To approach or exceed 99% in DRE it is necessary to use a synergistic combination of pre-test static analysis and inspections combined with formal testing using mathematically designed test cases, ideally created by certified test personnel. DRE can also be applied to defects found in other materials such as requirements and design. Table 20 illustrates current ranges for defect potentials and defect removal efficiency levels in the United States circa 2013 for applications in the 1000 function point size range:

Table 20: Function Points and Defect Removal Efficiency (DRE)

Best Case

Defect Origins	Defects per Function Point	Defect Removal Efficiency (DRE)	Delivered Defects per Function Point
Requirements	0.50	98.00%	0.01
Design	0.60	98.00%	0.01
Code	0.80	99.50%	0.00
User documents	0.40	99.00%	0.00
Bad Fixes	0.20	98.00%	0.00
TOTAL	**2.50**	**98.64%**	**0.03**

Average Case

Defect Origins	Defects per Function Point	Defect Removal Efficiency (DRE)	Delivered Defects per Function Point
Requirements	1.00	75.00%	0.25
Design	1.25	87.00%	0.16
Code	1.75	95.50%	0.08
User documents	0.60	91.00%	0.05
Bad Fixes	0.40	78.00%	0.09
TOTAL	**5.00**	**87.34%**	**0.63**

Worst Case

Defect Origins	Defects per Function Point	Defect Removal Efficiency (DRE)	Delivered Defects per Function Point
Requirements	1.50	70.00%	0.45
Design	2.00	80.00%	0.40
Code	2.50	92.00%	0.20
User documents	1.00	85.00%	0.15
Bad Fixes	0.75	68.00%	0.24
TOTAL	**7.75**	**81.42%**	**1.44**

Predictions of defect potentials and defect removal efficiency levels are standard features of Software Risk Master ™ (SRM). In fact not only are they standard, but they are also patent-pending features.

Function Points and Natural Metrics such as "Document Pages"

A function point is a synthetic metric comprised of five elements that are essentially invisible to the human eye or at least hard to see without close examination: inputs, outputs, inquiries, logical files, and interfaces.

A natural metric is a count of visible objects that are easy to see and in many cases can even be touched and examined using many senses. A prime example of a natural metric for software applications are "pages of documentation."

Software applications are highly paper driven. In fact some defense software projects create more than 100 document types containing more than 1,400 English words for every Ada statement. The cost of the words is greater than the costs of the code itself.

One of the interesting attributes of agile software development is a sharp reduction in paperwork volumes for requirements and design, due to having embedded users. Of course even agile cannot reduce paperwork for FDA or FAA certification, or for large defense contracts where production of various paper documents are contractually mandated.

Documentation is often produced in multiple languages. For example in Canada both French and English are required. For commercial products marketed globally user documents may need to be translated into 20 languages or more. Translation used to be a major cost element but automatic translation tools such as Google translate have lowered the cost.

Software documents are also major sources of error. Software requirements average about 1.0 defects per function point and design about 1.25 defects per function point. Summed together requirements and design defects often outnumber code defects, which average about 1.75 per function point.

Function point metrics are very useful for quantifying both the sizes of the various documents and also their costs for creation and updates. Document sizing using function point metrics is a standard feature of Software Risk Master ™ (SRM). This feature might have been patented when it was first developed, but the mathematics for sizing documents using function points was created by the author in the 1970's and hence is considered to be prior art. Table 20.1 illustrates a subset of total documentation for a major system of 10,000 function points in size:

Table 20.1: Function Points for Document Prediction
(Application of 10,000 function points in size)

Document Sizes	Pages	Pages per Funct. Pt.	English Words	English words Per Funct. Pt.	Percent Complete
Requirements	2,126	0.21	850,306	85.03	73.68%
Architecture	376	0.04	150,475	15.05	78.63%
Initial design	2,625	0.26	1,049,819	104.98	68.71%
Detail design	5,118	0.51	2,047,383	204.74	75.15%
Test plans	1,158	0.12	463,396	46.34	68.93%
Development Plans	550	0.06	220,000	22.00	76.63%
Cost estimates	376	0.04	150,475	15.05	79.63%
User manuals	2,111	0.21	844,500	84.45	85.40%
HELP text	1,964	0.20	785,413	78.54	86.09%
Courses	1,450	0.15	580,000	58.00	85.05%
Status reports	996	0.10	398,205	39.82	78.63%
Change requests	2,067	0.21	826,769	82.68	78.68%
Bug reports	11,467	1.15	4,586,978	458.70	81.93%
TOTAL	**32,384**	**3.24**	**12,953,720**	**1,295.37**	**78.24%**

Table 20.1 only illustrates a sample of major document types. (The full set of documents is too large for this paper.) Over and above the normal documentation shown in Table 20.1 quite a few special kinds of documents are needed for software projects that require FDA or FAA certification. Paperwork is a major software cost driver and function points are the best metric for quantifying both paperwork volumes and paperwork costs.

When you narrow the focus to a specific type of document, such as requirements, function point normalization reveals some important issues that could not easily be studied. Table 20.2 shows requirements for applications ranging in size from 10 to 100,000 function points:

Table 20.2: Initial Requirements Size and Completeness

Function Points	Require. Pages	Pages per Funct. Pt.	Complete. Percent	Days to Read	Amount Understood	Require. Defects
10	6	0.60	100.00%	0.10	100%	4
100	40	0.40	99.00%	0.68	100%	26
1,000	275	0.28	91.34%	5.02	93%	1 71
10,000	2,126	0.21	73.68%	48.09	13%	1,1 46
100,000	19,500	0.20	31.79%	600.00	2%	7,6 57

As projects grow in size requirements soon become too big for one person to read and understand. For a major system of 100,000 function points it would take 600 work days to read the requirements, and nobody could understand more than about 2% of them. This is why segmentation into smaller components is needed for major systems.

There are three key points for large software projects: 1) Paperwork is often the most expensive item produced on large software projects; 2) Some paper documents such as requirements and design contain many errors or defects that need to be included in quality studies; 3) Function point metrics are the most effective metric for sizing documents and studying document creation costs, as well as defects or bugs found in various documents.

For a 10,000 function point software application the U.S. average is about 10.43 pages per function points; 2,607 English words per function point. Total documentation costs are $783.86 per function point which comprises 24.30% of the total application development costs. Defects in requirements and design would top 2.25 per function point which is larger than code defects of about 1.75 defects per function point.

Function Points and Goal Question Metrics (GQM)

The goal, question, metric approach was developed by Dr. Victor Basili of the University of Maryland. It has become a useful and popular approach. The essential concept is to start with defining a business goal, then develop questions about how the goal might be approached, and then develop metrics that can measure the approach to the goal.

In its original form the GQM approach was highly individual and each company or application might have its own set of goals. However since software projects have been studied for more than 50 years they have a known set of major problems that would allow a standard set of goals and questions to be developed. Function point metrics are congruent with and can be used with

many of these standard software goals. A few examples of common problems and related goals might be:

- Reduce requirements creep > 0.5% per calendar month from today's rate of > 2.0% per calendar month.
- Reduce software defect potentials from > 5.00 defects per function point to < 2.50 defects per function point.
- Raise software defect removal efficiency up > 99% from today's value of < 86%.
- Improve customer satisfaction for released software to > 97% "satisfied" from today's value of < 80% satisfied.
- Reduce software development costs to < $400 per function point from today's average value of > $1000 per function point.
- Reduce annual maintenance costs to < $50 per function point from today's average value of > $125 per function point.
- Reduce cancellation rates for applications > 10,000 function points to < 1% from today's rate of > 15%.

These are general goals that deal with common software problems that occur with high frequency. There may also be unique and specific goals for individual projects, but all of the goals shown above probably need to be addressed for every major software project.

Function Points and Earned Value Analysis (EVA)

The earned-value method originated in the 1960's and has become a popular approach with its own professional association. EVA is used on many government and defense projects including both software and hardware. EVA is a large and complex topic and this paper only shows how function points can be congruent with other EVA measures.

The essence of EVA is that prior to starting a major project a development plan is created that shows progress of specific deliverables on a timeline. This is called Planned Value or PV. As the project gets under costs and results are tracked. Successful completion is called Earned Value of EV. If the project is running late or spending more than anticipated, the curves for PV and EV will draw apart, indicating that corrective actions are needed.

For software EVA is not a perfect tool because it omits quality, which is both a major cost driver for software and the #1 reason for schedule delays. A modified form of EVA for software would combine standard EVA tracking with an additional set of quality reports that compared planned defect removal efficiency (PDRE) with actual defect removal efficiency (ADRE). For example if the defect estimate for function test predicted 100 bugs and only 50 were found, it that due to the fact that quality was better than expected or due to the fact that defect removal efficiency was lower than expected?

Companies such as IBM with good defect prediction tools and sophisticated defect removal methods can easily modify EVA to include quality cost drivers. The current versions of Software

Risk Master ™ (SRM) predicts defects found by every form of removal activity including requirements and design inspections, static analysis, code inspections, and 18 different forms of testing. Independent verification and validation (IV&V) is also predicted for defense projects that require it.

Another use of function points in an EVA context would be to divide applications into discrete components. For example an application of 1,000 function points in size might be segmented into 10 components of 100 function points. Each component would be an EVA unit that could be inserted into standard EVA calculations.

Function Points, Story Points, and Velocity on Agile Projects

Function point metrics are not widely used on Agile projects, in part because manual counting of function points is too slow and expensive to fit the Agile philosophy. Many agile projects use story points as an alternate metric, and they use velocity as a tool for predicting completion of stories in a specific time period such as a week or a month.

The high-speed sizing method of Software Risk Master ™ (SRM) which sizes applications in about 1.8 minutes is a good match to the agile philosophy. To facilitate use by agile projects SRM can also do bi-directional conversion between story points and use case points. It can also predict velocity.

As a small example, assume an application of 1000 function points is being built using the agile methodology. That is roughly equivalent to 556 story points, assuming each user story encompasses about 2 function points. SRM predicts a total of 6 sprints for this project, each of which would last about 2.38 months. In total there would probably be about 71 scrum meetings. Each sprint would develop about 92 stories. Velocity would be about 39 stories per month.

Since SRM also has a very precise measurement mode, these predictions could be measured and changed by examining many agile projects. This is needed since unlike function points there is no certification for counting user stories and no ISO standard for what story points encompass. Among the author's clients story points vary by about 3 to 1 in contents. The bottom line is that function points can easily be added to the set of methods used by agile projects now that they can be calculated in only a few minutes. They would not replace story points but they would allow agile projects to be compared against large data bases such as those maintained by the International Software Benchmark Standards Group (ISBSG).

Function Points and Return on Investment (ROI)

To CEO's and other kinds of C-level executives return on investment or ROI is the top concern for every form of product development including software projects. The oldest methods of calculating ROI include accounting rates of return and internal rates of return. Newer methods include economic value added (EVA), return on assets (ROA), return on infrastructure employed (ROIE), and real options valuation (ROV). An interesting book that discusses these topics in a software context is The Business Value of IT by Michael Harris, David Herron, and Stasia Iwanicki. An older book that also contains useful value data is Software Project Management, A Unified Framework, by Walker Royce.

In general software applications provide value in one of three distinct fashions:

1. The software lowers operational costs and improves worker performance.
2. The software is marketed and generates both direct and indirect revenue streams.
3. The software provides intangible value such as aiding medical science or improving national security against external attack.

Since this paper is concerned with using function point metrics to demonstrate software economic value, only the first two methods will be discussed; operational efficiency and revenue generation.

Case 1: Software improves operational performance: Let us assume that an insurance company is building a claims management system that will improve claims handling by 20% compared to current methods.

Assume that the software is 10,000 function points in size and was developed at a cost of $1,000 per function point or $10,000,000 in total.

Assume the insurance company employees 1,000 claims agents and their compensation is $60,000 per year or a total of $60,000,000 per year.

A 20% improvement in agent performance will generate cost savings of $12,000,000 per year. If you assume that the internal time horizon for calculating ROI is 5 years, then the application would save $60,000,000 for a cost of $10,000,000. The simple ROI of the project would be $6.00 for every $1.00.

This is not necessarily an exciting ROI but it is good enough for the company to fund the project. Using function point metrics, the development cost of the application was $1,000 per function point and the value of the application over a 5-year period was $6,000 per function point.

This is a simple case to illustrate that function points are useful in value analysis. In real life maintenance, inflation, changes in numbers of workers, and many other factors would need to be included.

For example after the 20% annual increase in performance the company might decide to downsize claims agents and lay off 10% of the staff or 100 agents. This would change the long-range value calculations. Alternatively, the company's business might increase by 20% so the full current staff is still needed.

Case 2: Software generates direct and indirect revenues streams:

In this case let us assume that a new commercial software vendor funded by venture capital plans to bring out a new application that they expect will be used by many consumers or customers. Let us assume that the application is 1,000 function points in size and will be built at a cost of $1,000 per function point or $1,000,000 in total.

However since the company is new and venture funded, an additional $1,000,000 will be needed to fund marketing, sales, and management. A third $1,000,000 will be needed fund the advertising rollout for the application; i.e. the venture investment is $3,000,000 or $3,000 per function point.

The company assumes that this application will be acquired by 1,000,000 customers per year at a cost of $100 per copy, or $100,000,000 per year in direct revenues.

The company also assumes that 50% of the customers will request training in the application, which will be offered for $200 per client. Here too the annual revenues will be $100,000,000.

IF the project is delivered on time and meets expectations annual revenues will be $200,000,000.

This case has some caveats that need to be considered carefully:

- If the application is delivered one month early it might generate additional revenues of more than $16,000,000 for that month.
- If the application is delivered one month late it might lose revenues of more than $16,000,000 for the month.
- If the quality of the application is poor and defect removal efficiency is below 90% then potential sales will be reduced by 75%.
- If the quality of the application is high and defect removal efficiency is above 97% then potential sales will be increased by 25%.
- If the product is initially successful within a year "fast followers" will be offering similar products. The time line for zero competition is a very narrow band.

Venture capital horizons are normally only three years so let us consider what might occur in terms of value over a three-year period.

The **best-case scenario** is that by means of effective development utilizing inspections, static analysis, and formal testing the application is delivered 1 month early and has a measured defect removal efficiency level of 98%. Post-release maintenance will cost $500,000 per year.

In this case annual revenues will be $250,000,000 per year plus the extra $16,000,000 for being early. The three-year revenue stream would total to $786,000,000 which is $786,000 per function point.

When the initial venture investment if $3,000 per function point plus three years of maintenance total to $4,500,000. When this is compared to the three-year revenue stream the ROI is $175 for every $1.00 invested. This kind of ROI is the hope of every entrepreneur and every venture capitalist.

For the best-case scenario the total venture investment was $4,500 per function point and the revenue stream amounted to $786,000 per function point. The entrepreneurs are on their way to billionaire status. Patents and aggressive patent litigation keep fast followers at bay.

The **worst-case scenario** is that the company is inept in software development and tries to deliver early by bypassing pre-test static analysis and inspections to save time. Instead of saving time, the application is so buggy when test begins that the testing cycle stretches out for an extra 6 months so the application lost $96,000,000 in first year revenues.

Cumulative defect removal efficiency is a dismal 83%. Not only that but another round of venture funding is needed bringing the total investment to $6,000,000. Post-release maintenance on all of the delivered bugs cost $5,000,000 per year.

Due to shipping late and having poor quality the total revenues for three years are only $125,000,000. Between the initial investment of $6,000,000 and annual maintenance costs of $5,000,000 per year the three-year costs for this case are $21,000,000.

If you divide the revenues of $125,000,000 by the costs of $21,000,000 the ROI for the project is positive but only $5.95 for every dollar invested. This low ROI is actually below the level that most venture funds would invest in.

The three year costs totaled to an alarming $21,000 per function point. The revenue stream was only $125,000 per function point. While the company managed to stay in business for three years, which is rare for venture-funded software companies, it is obvious that poor quality is reducing market share and raising maintenance costs to unacceptable levels.

Worse, the basic idea of the product attracted the attention of “fast followers” who bring out similar products with extra features at lower cost and better quality. As a result revenues erode and market share plummets.

Within two years the initial product is generating only $10,000,000 per year in revenues with expenses of $10,000,000 due to poor quality. The company goes bankrupt in year 5 while the leading fast follower establishes a strong new market for the basic idea and soon becomes a billion-dollar company.

The essential point is that function point metrics are useful in value analysis, but in order to optimize value and make software appealing to CEO's, other C-level executives, and to the venture capital community the software organization needs to understand effective development practices and also software economics.

Partial measures such as "design, code, and unit test" or DCUT have no place in economic value analysis, nor do inaccurate measures such as "lines of code" (LOC), and "cost per defect." Leakage or failing to measure 100% of development costs should also be avoided.

The recent "technical debt" metric is an interesting and useful metaphor, but woefully incomplete. As currently defined by a majority of users "technical debt" only covers about 17% of the total cost of poor quality.

Technical debt omits the costs of software projects whose quality is so bad they are canceled and never released. Even more serious, technical debt omits the costs of litigation and damages against vendors who are sued for poor quality and lose the lawsuits. The costs of litigation and damages can be larger than normal "technical debt" costs by more than 1000 to 1.

For that matter expensive and ineffective development methods such as pair programming should be avoided. Pair programming more than doubles software costs at no tangible improvement in schedules, quality, or application value.

Function point metrics can be applied to software size, software development, software documentation, software quality, software maintenance, software outsource contracts, and software venture capital investment. No other software metric has such a wide range of usefulness.

Summary and Conclusions

Function point metrics are the most powerful metrics yet developed for studies of software economics, productivity, risks, and quality. They are much better than older metrics such as "lines of code" and "cost per defect." They are also much better than alternate metrics such as "story points" and "use-case points".

However the slow speed and high costs of manual function point analysis has caused function points to be viewed by top-executives such as CEO's as a minor niche metric. In order to be useful to C level executives function point metrics need:

1. Faster counting by more than an order of magnitude from today's averages
2. Lower costs down below $0.05 per function point counted
3. Methodology benchmarks for all known methods
4. Quality benchmarks for defect prevention, pre-test removal and testing
5. Maintenance, enhancement, and total cost of ownership (TCO) benchmarks
6. Portfolio benchmarks for many companies and government groups
7. Industry benchmarks for all software-intensive industries
8. Global benchmarks for all countries that produce software in large volumes

This paper discusses a patent-pending method of sizing software projects in less than 2 minutes, with function points as one of the default metrics produced. Software Risk Masters ™ (SRM) produces development estimates in about 3 minutes; quality estimates in 4 minutes; and maintenance estimates in 4 minutes.

The Software Risk Master ™ tool can do more than size. The SRM tool can also predict the results and risks of any methodology, any level of team experience, any CMMI level, and programming language (or combination), and any volume of reusable materials.

Function point metrics might well become the global standard for software economic analysis once they can be applied to large systems > 10,000 function points; use full activity-based costs instead of partial project data; are applied to portfolios which might contain thousands of applications and millions of function points; and to industry and national software comparisons. There are no other metrics that are as effective as function points for software economic analysis.

References and Readings

Jones, Capers; "A Short History of Lines of Code Metrics"; Namcook Analytics Technical Report; Narragansett, RI; 2012.

This report provides a mathematical proof that "lines of code" metrics violate standard economic assumptions. LOC metrics make requirements and design invisible. Worse, LOC metrics penalize high-level languages. The report asserts that LOC should be deemed professional malpractice if used to compare results between different programming languages. There are other legitimate purposes for LOC, such as merely measuring coding speed.

Jones, Capers; "A Short History of the Cost Per Defect Metrics"; Namcook Analytics Technical Report; Narragansett, RI 2012.

This report provides a mathematical proof that "cost per defect" penalizes quality and achieves its lowest values for the buggiest software applications. It also points out that the urban legend that "cost per defect after release is 100 times larger than early elimination" is not true. The reason for expansion of cost per defect for down-stream defect repairs is due to ignoring fixed costs. The cost per defect metric also ignores many economic topics such as the fact that high quality leads to shorter schedules.

Jones, Capers: "*Sizing Up Software*;" Scientific American Magazine, Volume 279, No. 6, December 1998; pages 104-111.

Jones, Capers; "Early Sizing and Early Risk Analysis"; Capers Jones & Associates LLC; Narragansett, RI; July 2011.

Jones, Capers and Bonsignour, Olivier; The Economics of Software Quality; Addison Wesley Longman, Boston, MA; ISBN 10: 0-13-258220—1; 2011; 585 pages.

Jones, Capers; Software Engineering Best Practices; McGraw Hill, New York, NY; ISBN 978-0-07-162161-8; 2010; 660 pages.

Jones, Capers; Applied Software Measurement; McGraw Hill, New York, NY; ISBN 978-0-07-150244-3; 2008; 662 pages.

Jones, Capers; Estimating Software Costs; McGraw Hill, New York, NY; 2007; ISBN-13: 978-0-07-148300-1.

Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA; ISBN 0-201-48542-7; 2000; 657 pages.

Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Software Productivity Research, Inc.; Burlington, MA; September 2007; 53 pages; (SPR technical report).

Additional Literature

The literature on function point metrics is quite extensive. Following are some of the more useful books:

Abran, Alain and Dumke, Reiner R; Innovations in Software Measurement; Shaker-Verlag, Aachen, DE; ISBN 3-8322-4405-0; 2005; 456 pages.

Abran, Alain; Bundschuh, Manfred; Dumke, Reiner; Ebert; Christof; and Zuse, Horst; *Software Measurement News*; Vol. 13, No. 2, Oct. 2008 (periodical).

Bundschuh, Manfred and Dekkers, Carol; The IT Measurement Compendium; Springer-Verlag, Berlin, DE; ISBN 978-3-540-68187-8; 2008; 642 pages.

Chidamber, S.R. & Kemerer, C.F.; "*A Metrics Suite for Object-Oriented Design*"; IEEE Trans. On Software Engineering; Vol. SE20, No. 6; June 1994; pp. 476-493.

Dumke, Reiner; Braungarten, Rene; Büren, Günter; Abran, Alain; Cuadrado-Gallego, Juan J; (editors); Software Process and Product Measurement; Springer-Verlag, Berlin; ISBN 10: 3-540-89402-0; 2008; 361 pages.

Ebert, Christof and Dumke, Reiner; Software Measurement: Establish, Extract, Evaluate, Execute; Springer-Verlag, Berlin, DE; ISBN 978-3-540-71648-8; 2007; 561 pages.

Gack, Gary; Managing the Black Hole: The Executives Guide to Software Project Risk; Business Expert Publishing, Thomson, GA; 2010; ISBN10: 1-935602-01-9.

Gack, Gary; *Applying Six Sigma to Software Implementation Projects*; http://software.isixsigma.com/library/content/c040915b.asp.

Galorath, Dan and Evans, Michael; Software Sizing, Estimation, and Risk Management; Auerbach Publications, Boca Raton, FL; 2006.

Garmus, David & Herron, David; Measuring the Software Process: A Practical Guide to Functional Measurement; Prentice Hall, Englewood Cliffs, NJ; 1995.

Garmus, David and Herron, David; Function Point Analysis – Measurement Practices for Successful Software Projects; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3;363 pages.

Gilb, Tom and Graham, Dorothy; Software Inspections; Addison Wesley, Reading, MA; 1993; ISBN 10: 0201631814.

Harris, Michael D.S., Herron, David, and Iwanicki, Stasia; The Business Value of IT; CRC Press, Auerbach; Boca Raton, FL; 2008; ISBN 978-14200-6474-2.

International Function Point Users Group (IFPUG); IT Measurement – Practical Advice from the Experts; Addison Wesley Longman, Boston, MA; 2002; ISBN 0-201-74158-X; 759 pages.

Kemerer, C.F.; "*Reliability of Function Point Measurement - A Field Experiment*"; Communications of the ACM; Vol. 36; pp 85-97; 1993.

Parthasarathy, M.A.; Practical Software Estimation – Function Point Metrics for Insourced and Outsourced Projects; Infosys Press, Addison Wesley, Upper Saddle River, NJ; 2007; ISBN 0-321-43910-4.

Putnam, Lawrence H.; Measures for Excellence -- Reliable Software On Time, Within Budget; Yourdon Press - Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-567694-0; 1992; 336 pages.

Putnam, Lawrence H and Myers, Ware.; Industrial Strength Software - Effective Management Using Measurement; IEEE Press, Los Alamitos, CA; ISBN 0-8186-7532-2; 1997; 320 pages.

Royce, Walker; Software Project Management – Unified Framework; Addison Wesley, Boston, MA; 1999.

Stein, Timothy R; The Computer System Risk Management Book and Validation Life Cycle; Paton Press, Chico, CA; 2006; ISBN 10: 1-9328-09-5; 576 pages.

Stutzke, Richard D; Estimating Software-Intensive Systems; Addison Wesley, Upper Saddle River, NJ; 2005; ISBN 0-201-70312-2; 918 pages.