A PROPOSED SUITE OF THIRTEEN FUNCTIONAL METRICS FOR ECONOMIC ANALYSIS

Capers Jones, VP and CTO, Namcook Analytics LLC

Version 7.0 September 16, 2015

Abstract

Function point metrics are the most accurate and effective metrics yet developed for performing software economic studies, quality studies, and value analysis. The success of function point metrics for software applications leads to the conclusion that the logic of function point metrics should be applied to a linked suite of similar metrics that can size other business and technical topics.

This article suggests that an integrated suite of functional metrics be created that would encompass not only software via function points but also data points, risk points, value points, service points, web-site points, security points, hardware function points, and software usage points.

The reason for this suggestion is to enable large-scale economic analysis of complex systems that involve software, data, hardware, web sites, and other business topics that need concurrent sizing, planning, estimating, and economic analysis.

ECONOMIC ANALYSIS USING A SUITE OF FUNCTIONAL METRICS

INTRODUCTION

From their first publication in 1978 function point metrics have proven their value for software application sizing, cost estimation, quality predictions, benchmarks, and overall economic studies.

The International Function Point Users Group (IFPUG) has become the largest software measurement association in the world. There are also other function point variants that are growing rapidly too, including COSMIC function points, NESMA function points, FISMA function points, and a number of others.

Yet software does not exist in a vacuum. There are many related business topics which lack effective size metrics. One critical example is that of the data used by software applications.

Most large companies own more data than they do software. The costs of acquiring data and maintaining it are at least as high as software development and maintenance costs. Data migration from legacy applications to new applications can take more than three calendar years. Data quality is suspected to be worse than software quality, but no one really knows because there is no effective size metric for quantifying data-base volumes or measuring data quality.

It would seem to be useful to apply the logic of function point metrics to other critical business topics, and create an integrated suite of functional metrics that could encompass not only software, but the related areas of data, web sites, hardware devices, and also risk and value.

Software and on-line data are among the most widely utilized commodities in human history. If you consider the total usage of various commodities, the approximate global rank in terms of overall usage would be:

1. Water
2. Salt
3. Rice
4. Wheat
5. Bread
6. Corn
7. Fish
8. Clothing
9. Shoes
10. Software
11. On-line web data
12. Alcoholic beverages
13. Electricity

14. Gasoline and oil
15. Aluminum

(The sources of data for this table include a number of web sites and government tables. The importance is not actual rankings, but the fact that software and on-line data in 2015 are used so widely that they can be included in the list.)

The expansion of software (and on-line data) to join the world's most widely used commodities means that there is an urgent need for better metrics and better economic analysis.

Because of the widespread deployment of software and the millions of software applications already developed or to be developed in the future, software economic studies are among the most critical of any form of business analysis. Unfortunately, lack of an integrated suite of metrics makes software economic analysis extremely difficult.

This article proposes a suite of related metrics that are based on the logic of function points, but expanding that logic to other business and technical areas. The metrics are hypothetical and additional research would be needed to actually develop such a metrics suite.

POTENTIAL EXPANSION OF FUNCTIONAL METRICS TO OTHER TOPICS

In spite of the considerable success of function point metrics in improving software quality and economic research, there are a number of important topics that still cannot be measured well or even measured at all in some cases. Here are some areas where there is a need for of related metrics within a broad family of functional metrics:

1. Application function point metrics
2. Component feature point metrics
3. Hardware function point metrics
4. COTS application point metrics
5. Micro function point metrics
6. Data point metrics
7. Web-site point metrics
8. Software usage point metrics
9. Service point metrics
10. Risk point metrics
11. Value point metrics
12. Security point metrics
13. Configuration point metrics (developed by IBM)

This combination of a related family of functional metrics would expand the ability to perform economic studies of modern businesses and government operations that use

software, web sites, data, and other business artifacts at the same time for the same ultimate goals. Let us now consider each of these metrics in turn.

The Need for Application Function Point Metrics

From their first external publication outside of IBM in 1978 function point metrics have become the de facto standard for quantifying software applications. As of 2015 the usage of function points encompass international benchmark studies, outsource agreements, economic analysis, quality analysis, and many other important business topics. In 2015 the governments of Brazil, Japan, Malaysia, Mexico, and South Korea now require function point metrics for software contracts. The major topics found within function points as originally defined by Allan Albrecht include:

- **Function Points**
  Inputs
  Outputs
  Inquires
  Logical files
  Interfaces
  Complexity adjustments

There are a number of tools available for counting function points, but human judgment is also needed. Both IFPUG and the other major function point user groups provide training and also examinations that lead to the position of "certified function point analysts".

Function points are now the most widely used metric for quantifying software application size, for quantifying productivity and quality, and for quantifying application development costs. There is only sparse data on application maintenance costs, but that situation is improving. The International Software Benchmark Standards Group (ISBSG) now includes software maintenance data. Several companies such as the Software Improvement Group (SIG), Relativity Technologies, CAST Software, Optimyth, and Computer Aid measure and evaluate maintainability.

It should be noted that software is treated as a taxable asset by the Internal Revenue Service (IRS) in the United States and by most other international tax organizations. Function point metrics are now widely used in determining the taxable value of software when companies are bought or sold.

Note 1: In late 2011 the International Function Point Users Group (IFPUG) released information on a new metric for non-functional requirements called SNAP. As of early 2015 there is still sparse empirical data on the volume of SNAP points relative to normal function points in the same application. As data becomes available it will be added to software estimating tools such as the author's Software Risk Master ™ tool.

Also function point metrics have a tendency to be troublesome for maintenance and multi-tier software where quite a bit of work involves dealing with surrounding software packages.

Note 2: This paper is based on function points as defined by IFPUG. There are a number of alternative function point metrics including but not limited to:

COSMIC function points
Engineering function points
FISMA function points
Mark II function points
NESMA function points
Unadjusted function points
Story points
Use case points

These function point variations all produce different results from IFPUG function points. Most produce larger results than IFPUG for unknown reasons.

The Need for Component Feature Point Metrics

While function points are the dominant metric for software applications, in today's world of 2015 applications are often created from libraries of reusable components, objects, and other existing software segments. While some of these may have been counted via normal function point analysis, most are of unknown size.

There is a need to extend normal function point analysis down at least one level to be able to size reusable modules, objects, and the contents of class libraries. To avoid confusion with the term function points, which normally apply to entire applications, it might be better to use a different term such as "component feature points."

- **Component Feature Points**
  Inputs
  Outputs
  Inquires
  Logical files
  Interfaces
  Complexity adjustments

Examples of the kinds of specific features that might be sized using component feature points would include, but not be limited to:

1. Input validation (25 to 50 component feature points)
2. Output formatting (10 to 30 component feature points)
3. Query processing (3 to 15 component feature points)
4. Currency exchange rate calculation (5 to 15 component feature points)

5. Inflation rate calculation (5 to 10 component feature points)
6. Compound interest calculation (5 to 25 component feature points)
7. Sensor-based input monitoring (10 to 35 component feature points)
8. Earned-value calculations (30 to 75 component feature points)
9. Internal rate of return (IRR) (5 to 15 component feature points)
10. Accounting rate of return (ARR) (5 to 15 component feature points)

The basic idea is to assemble a taxonomy of standard components that are likely to be acquired from reusable sources rather than custom developed. In other words, component feature points shift the logic of functional analysis from the external applications themselves to the inner structure and anatomy of applications.

As of 2015 the total number of possible reusable components is unknown, but probably is in the range of about 500 to 2,500. There is also a lack of a standard taxonomy for identifying the specific features of software components. These are problems that need additional research.

The best way to develop an effective taxonomy of application features would probably be a forensic analysis of a sample of current software applications, with the intent of establishing a solid taxonomy of specific features including those inserted from reusable materials.

Component feature points would adhere to the same general counting rules as standard function points, but would be aimed at individual modules and features that are intended to be reused in multiple applications. Because some of the smaller components may be below the boundary line for normal function point analyses, see the section on "micro function points" later in this paper.

The Need for Hardware Function Point Metrics

The U.S. Navy, the U.S. Air Force, the U.S. Army and the other military services have a significant number of complex projects that involve hardware, software, and microcode. Several years ago the Navy posed an interesting question: "Is it possible to develop a metric like function points for hardware projects, so that we can do integrated cost analysis across the hardware/software barrier?"

In addition to military equipment there are thousands of products that feature embedded software: medical devices, smart phones, GPS units; cochlear implants, hearing aids, pacemakers, MRI devices, automobile anti-lock brakes; aircraft control systems, and countless others. All of these hybrid devices require sizing and estimating both the software and hardware components at the same time.

The ability to perform integrated sizing, cost, and quality studies that could deal with software, hardware, data bases, and human service and support activities would be a notable advance indeed. A hypothetical engineering point metric might include the following factors:

- **Hardware Function points**
  - Inputs
  - Outputs
  - Constraints
  - Innovations
  - Algorithms
  - Subcomponents

Integrated cost estimates across the hardware/software boundary would be very welcome in many manufacturing and military domains. These hardware function points would be utilized for embedded applications such as medical devices, digital cameras, and smart appliances. They would also be used for weapons systems and avionics packages. They would also be used for all complex devices such as automobile engines that use software and hardware concurrently. Hardware function points would be a useful addition to an overall metrics suite.

The Need for COTS Function Point Metrics

Many small corporations and some large ones buy or acquire more software than they build. The generic name for packaged applications is "commercial off-the-shelf software" which is usually abbreviated to COTS.

COTS packages could be sized using conventional function point analysis if vendors wished to do this, but most do not. As of 2015 it is technically possible to size COTS packages using pattern matching. For example the Software Risk Master (SRM) sizing tool of Namcook Analytics LLC can size COTS software such as Windows 10, Quicken, the Android operating system and all others. The same is true for sizing open-source applications. The open-source business sector is growing rapidly, and many open-source applications are now included in corporate portfolios.

The concept of pattern matching uses a formal taxonomy of applications types that includes the class of the application (internal or external), the type (embedded software, information technology, systems or middleware, etc.) and several other parameters. An application to be sized is placed on the taxonomy. Applications that have the same "pattern" on the taxonomy are usually of almost the same size in function points. The pattern matching approach uses a combination of a standard taxonomy and mathematical algorithms to provide a synthetic function point total, based on historical applications whose sizes already exist. While normal function points are in the public domain, the pattern matching approach is covered by a patent application. Some of the other metrics in this paper may also include patentable algorithms.

The pattern matching approach substitutes historical data for manual counting, and to be effective the patterns must be based on a formal taxonomy. Pattern matching applies some of the principles of biological classification to software classification.

A study performed by the author of the corporate portfolio of a major Fortune 500 corporation noted that the company owned software in the following volumes:

Application Types	**Ownership**
Information systems	1,360
COTS packages	1,190
Systems software	850
Embedded applications	510
Tools (software development)	340
Manufacturing and robotics	310
End-user developed	200
Open-source	115
SaaS applications	5
TOTAL	4,880

As can be seen, COTS packages ranked number two in the corporation's overall portfolio and comprised 24.4% of the total portfolio. This is far too important a topic to be excluded from sizing and economic analysis. For one thing, effective "make or buy" analysis or determining whether to build software or acquire software packages needs the sizes of both the COTS packages and the internal packages to ensure that features sets are comparable. In fact both function points and component feature points would be valuable for COTS analysis.

Note that the pattern-matching method can also size "Software as a Service" or SaaS applications such as Google Docs. Essentially any software application can be sized using this method so long as it can be placed on the basic taxonomy of application types. Of course the complexity questions will have to be approximated by the person using the sizing method, but most can be assumed to center on "average" values.

Examples of various COTS, SaaS, and open-source applications sized via pattern matching include:

Table 1: Examples of Software Size via Pattern Matching Using Software Risk Master ™

Application	**Size in IFPUG Function Points**
1. Oracle	229,344
2. Windows 10	198,050
3. Microsoft Windows XP	126,768
4. Microsoft Office 2010	93,498
5. Google docs	47,668
6. Apple I Phone	19,366
7. IBM IMS data base	18,955

8.	Google search engine	18,640
9.	Linux	17,505
10.	Child Support Payments (state)	12,546
11.	Facebook	8,404
12.	Mapquest	3,793
13.	Android OS (original version)	1,858
14.	Microsoft Excel	1,578
15.	Microsoft Word	1,431
16.	Laser printer driver (HP)	1,248
17.	Sun Java compiler	1,185
18.	Wikipedia	1,142
19.	Cochlear implant (embedded)	1,041
20.	Microsoft DOS circa 1998	1,022
21.	Nintendo Gameboy DS	1,002
22.	Casio atomic watch	933
23.	SPR KnowledgePlan	883
24.	Norton anti-virus	700
25.	Golf handicap analysis	662
26.	SPR SPQR/20	699
27.	Google Gmail	590
28.	Cochlear implant (embedded)	546
29.	Twitter (original circa 2009)	541
30.	Software Risk Master™ prototype 1	388

Right now, COTS packages and SaaS packages (and most open-source applications) are outside the boundaries of normal function point metrics primarily because the essential inputs for function point analysis are not provided by the vendors.

It would be useful to include COTS packages in economic studies if vendors published the function point sizes of commercial software applications. This is unlikely to happen in the near future. A COTS, SaaS, and open-source pattern-matching metric based on pattern matching might include the following factors:

- **COTS, SaaS, and Open-Source application points**
  Taxonomy
  Scope
  Class
  Type
  Problem complexity
  Code complexity
  Data complexity

The inclusion of COTS points is desirable for dealing with "make or buy" decisions in which possible in-house development of software is contrasted with possible acquisition of a commercial package.

In today's world many large and important applications are combinations of custom code, COTS packages, open-source packages, reusable components, and objects. There is a strong business need to be able to size these hybrid applications.

There is also a strong business need to be able to size 100% of the contents of corporate portfolios, and almost 50% of the contents of portfolios are in the form of COTS packages, open-source packages, SaaS services and other kinds of applications whose developers have not commissioned normal function point analysis.

The Need for Micro Function Point Metrics

A surprising amount of software work takes place in the form of very small enhancements and bug repairs that are below about 10 function points in size. In fact almost 20% of the total effort devoted to software enhancements and about 90% of the effort devoted to software bug repairs deal with small segments below 10 function points in size.

The original function point metric had mathematical limits associated with the complexity adjustment factors which made small applications difficult to size. Also, the large volume of small enhancements and the even larger volume of software defect repairs would be time consuming and expensive for normal function point analysis.

The same method of pattern matching can easily be applied to small updates and bug repairs, and this form of sizing takes only a few minutes.

There are three possibilities for micro function points: 1) Normal function point analysis with changes to eliminate the lower boundaries of adjustment factors; 2) Pattern matching; 3) Backfiring or mathematical conversion from counts of logical code statements.

- **Micro Function Points using normal counts**
  Inputs
  Outputs
  Inquires
  Logical files
  Interfaces
  Revised complexity adjustments

- **Micro Function Points using pattern matching**
  Taxonomy
  Scope
  Class
  Type
  Problem complexity
  Code complexity
  Data complexity

- **Backfiring**

Language Level	Sample Languages	Source code per function point
1	Basic Assembly	320
2	C	160
3	COBOL	107
4	PL/I	80
5	Ada95	64
6	Java	53
7	Ruby	46
8	Oracle	40
9	Pearl	36
10	C++	32
11	Delphi	29
12	Visual Basic	27
13	ASP NET	25
14	Eiffel	23
15	Smalltalk	21
16	IBM ADF	20
17	MUMPS	19
18	Forte	18
19	APS	17
20	TELON	16
10	AVERAGE	58

Backfiring or mathematical conversion from logical code statements is as old as function point analysis. The first backfire results were published by Allan Albrecht in the 1970's based on simultaneous measurements of logical code statements and function points within IBM.

Surprisingly, none of the function point organizations have ever analyzed backfire data. Backfiring is not as accurate as normal function point analysis due to variations in programming styles but it remains a popular method due the high speed and low cost of backfiring compared to normal function point analysis..

There are published tables of ratios between logical code statements and function points available for about 800 programming languages. In fact the number of companies and projects that use backfiring circa 2011 is probably larger than the number of companies that use normal function point analysis.

As an example of why micro function points are needed, a typical software bug report when examined in situ in the software itself is usually between about 0.1 and 4.0 function points in size: much too small for normal function point analysis.

Individually each of these bugs might be ignored, but large systems such as Windows 7 or SAP can receive more than 50,000 bug reports per year. Thus the total volume of these tiny objects can top 100,000 function points and the costs associated with processing them can top $50,000,000 per year. There is a definite need for a rapid and inexpensive method for including thousands of small changes into overall software cost and economic analyses.

Since normal function point analysis tends to operate at a rate of about 400 function points per day or 50 function points per hour, counting a typical small enhancement of 10 function points would require perhaps 12 minutes.

The pattern matching method operates more or less at a fixed speed of about 1.5 minutes per size calculation, regardless of whether an ERP package of 300,000 function points or an enhancement of 10 function points is being sized. Therefore pattern matching would take about 1.5 minutes.

What would probably be a suitable solution would be to size a statistically valid sample of several hundred small bug repairs and small enhancements, and then simply use those values for sizing purposes. For example if an analysis of 1000 bugs finds the mean average size to be 0.75 function points that value might be used for including small repairs in overall economic studies.

It might be noted that the author's Software Risk Master ™ tool can size applications over a range that spans from less than 1 function point to more than 300,000 function points. Further, the time required to size the application is independent of the actual size and averages about 1 minute and 30 seconds per application.

The Need for Data Point Metrics

In addition to software, companies own huge and growing volumes of data and information. As topics such as repositories, data warehouses, data quality, data mining, and on-line analytical processing (OLAP) become more common, it is obvious that there are no good metrics for sizing the volumes of information that companies own. Neither are there good metrics for exploring data quality, the costs of creating data, migrating data, or eventually retiring aging legacy data.

A metric similar to function points in structure but aimed at data and information rather than software would be a valuable addition to the software domain. A hypothetical data point metric might include the following factors:

- **Data points**
  Logical files
  Entities
  Relationships
  Attributes
  Inquiries

Interfaces

Surprisingly, data base and data warehouse vendors have performed no research on data metrics. Each year more and more data is collected and stored, but there are no economic studies of data costs, data quality, data life expectancy, and other important business topics involving data.

If you look at the entire portfolio of a major corporation such as large bank, they probably own about 3,000 software applications with an aggregate size of perhaps 7,500,000 function points. But the volume of data owned by the same bank would probably 50,000,000 data points, if there were an effective data point metric in existence.

It is a known fact that the average number of software defects released to customers in 2015 is about 0.45 per function point. No one knows the average number of data errors, but from analysis of data problems within several large companies, it is probable that data errors in currently active data bases approach 2.5 defects per "data point" or almost four times as many errors as software itself.

There is a very strong economic need to include data acquisition costs, data repair costs, and data quality in corporate financial analyses. The data point metric would be probably as useful and as widely utilized as the function point metric itself. Lack of quantification of data size, data acquisition costs, data migration costs, and data quality are critical gaps in corporate asset economic analysis. A data point is important enough so that it might well be protected by a patent.

Data is already a marketable product and hundreds of companies sell data in the form of mailing lists, financial data, tax information and the like. If data is treated as a taxable asset by the Internal Revenue Service (IRS) then the need for a data point metric will be critical for tax calculations, and for use in determining the asset value of data when companies are bought or sold.

Since the theft of valuable data is now one of the most common crimes in the world, an effective data point metric could also be used in ascertaining the value of lost or stolen data.

The Need for Web-Site Point Metrics

In today's business world of 2015 every significant company has a web site, and an ever-growing amount of business is transacted using these web sites.

While function points can handle the software that lies behind the surface of web sites, function points do not deal with web site content in the forms of graphical images, animation, and other surface features. There is a strong business need to develop "web site points" that would be able to show web site development costs, maintenance costs, and web site quality.

Some of the topics that would be included in "web-site points" would be:

- **Web-site points**
  Transactions
  Inquiries
  Images
  Text
  Audio
  Animation

An examination of any of today's large and complex web sites, such as Amazon, Google, state governments, and even small companies immediately demonstrates that sizing and quantification are needed for many more topics than just the software that controls these web sites.

From a rudimentary analysis of web-site economics, it appears that the cost of the content of web sites exceeds the cost of the software controlling the web site by somewhere between 10 to 1 and 100 to 1. Massive web sites such as Amazon are at the high-end of this spectrum. But the essential point is that web sites need formal sizing methods and reliable economic methods.

The software that controls the Amazon web site is probably about 18,000 function points in size. But the total web content displayed on the Amazon site would probably top 25,000,000 web-site points if such a metric existed.

The Need for Software Usage Point Metrics

Function point metrics in all of their various flavors have been used primarily to measure software development. But these same metrics can also be used to measure software usage and consumption.

In order to come to grips with software usage patterns, some additional information is needed:

Is the software used by knowledge workers such as physicians and lawyers?
Is the software used for business transactions such as sales?
Is the software used to control physical devices such as navigational instruments?
If the software used to control military weapons systems?

Table 1.0 illustrates the approximate usage patterns noted for 30 different occupation groups last year in 2015:

Table 1: Daily Software Usage by Thirty Occupation Groups
(Size expressed in terms of IFPUG function points, version 4.2)

	Occupation Groups	Size in Function Points	Number of Packages	Hours used per Day	Value to Users
1	NSA analysts	7,500,000	60	24.00	10.00
2	Military planners	5,000,000	50	7.50	9.00
3	Astronaut (space shuttle)	3,750,000	50	24.00	10.00
4	Physicians	3,500,000	25	3.00	9.00
5	Ship captains (naval)	2,500,000	60	24.00	8.00
6	Aircraft pilots (military)	2,000,000	50	24.00	10.00
7	FBI Agents	1,250,000	15	3.00	7.00
8	Ship captains (civilian)	1,000,000	35	24.00	7.00
9	Biotech researchers	1,000,000	20	4.50	6.00
10	Airline pilots (civilian)	750,000	25	12.00	7.00
11	Movie special effects engineer	750,000	15	6.00	9.00
12	Air-traffic controllers	550,000	5	24.00	9.00
13	Attorneys	325,000	12	2.50	5.00
14	Combat officers	250,000	12	10.00	6.00
15	Accountants	175,000	10	3.00	4.00
16	Pharmacists	150,000	6	3.50	4.00
17	U.S. congress staff	125,000	15	6.00	4.00
18	Electrical engineers	100,000	25	2.50	5.00
19	Combat troops	75,000	7	18.00	6.00
20	Software engineers	50,000	20	6.50	8.00
21	Police officers	50,000	6	8.00	4.00
22	Corporate officers	50,000	10	1.50	3.00
23	Stock brokers	50,000	15	10.00	5.00
24	Project managers	35,000	15	2.00	5.00
25	IRS tax agents	35,000	12	8.00	6.00
26	Civil engineers	25,000	10	2.00	6.00
27	Airline travel reservations	20,000	3	12.00	9.00
28	Railroad routing and control	15,000	3	24.00	9.00
29	Customer support (software)	10,000	3	8.00	4.00
30	Supermarket clerks	3,000	2	7.00	4.00
	Averages	1,036,433	20	10.88	6.60

Software usage points are identical to normal function points, except that they are aimed at consumption of software rather than production of software. Software usage patterns play a major role in quantifying the value of many software applications. Software usage can be calculated using either normal function point analysis or pattern matching.

- **Usage Points using normal function point counts**
  Inputs
  Outputs
  Inquires

Logical files
Interfaces
Revised complexity adjustments
Knowledge usage
Operational usage
Transactional usage
Indirect usage (in embedded devices)

- **Usage Points using pattern matching**
  Taxonomy
  Scope
  Class
  Type
  Problem complexity
  Code complexity
  Data complexity
  Knowledge usage
  Operational usage
  Transactional usage
  Indirect usage (in embedded devices)

Usage points are not really a brand new metric but rather function points augmented by additional information and aimed in a different direction.

Incidentally it is from examining software usage patterns that led to placing software as number 10 on the list of widely-used commodities at the beginning of this article.

The Need for Service Point Metrics

The utility of function points for software studies has raised the question as to whether or not something similar can be done for service groups such as customer support, human resources, sales personnel, and even health and legal professionals.

Once software is deployed, a substantial amount of effort is devoted to responding to customer request for support. This service effort consists of answering basic questions, dealing with reported bugs, and making new information available to clients as it is created.

The cost drivers of software service are based on five primary factors:

1. The size of the application in function points
2. The number of latent bugs in the application at release
3. The number of clients using the application
4. The number of translations into other national languages
5. The planned response interval for customer support contacts

What would be useful would be a metric similar in structure to function points, only aimed at service functions within large corporations. Right now, there is no easy way to explore the lifetime costs of systems that include extensive human service components as well as software components. A hypothetical service point metric might include the following factors:

- **Service points**
  - Customers (entities)
  - Countries where the application is used
  - Latent defects at deployment
  - Desired response time for customer contacts
  - Inquiries
  - Reference sources
  - Rules and Regulations (constraints)

Experiments with variations on the function point metric have been carried out for software customer support groups. The results have been encouraging, but are not yet at a point for formal publication.

The U.S. is now largely a service-oriented economy. Software has a significant amount of total cost of ownership tied up in service-related activities.

The Need for Value Point Metrics

One of the major weaknesses of the software industry has been in the area of value analysis and the quantification of value. All too often what passes for "value" is essentially nothing more than cost reductions or perhaps revenue increases. While these are certainly important topics, there are a host of other aspects of value that also need to be examined and measured: customer satisfaction, employee morale, national security, safety, medical value, and a host of other topics. A hypothetical value point metric might include the following factors:

- **Value points**
  - Safety improvement
  - National security improvement
  - Health and medical improvement
  - Patents and intellectual property
  - Risk reduction
  - Synergy (compound values)
  - Cost reduction
  - Revenue increases
  - Market share increases
  - Schedule improvement
  - Competitive advantages
  - Customer satisfaction increase
  - Staff morale increase

Mandates or statutes

Note that although cost reduction and revenue increases are both tangible value factors, a host of other less tangible factors also need to be examined, weighted, and included in a value point metric.

Intangible value is the current major lack of today's methods of value analysis. There is no good way to quantify topics such as medical value, security value, or military value.

A value point metric would assign points for: 1) Direct revenues; 2) Indirect revenues; 3) Transaction rate improvements; 4) Operational cost reduction; 5) Secondary cost reduction; 6) Patents and intellectual property; 7) Enterprise prestige; 8) Market share improvements, 9) Customer satisfaction improvement; 10) Employee morale improvements. In other words both financial and non-financial value would be assigned value points. The sum total of value points would include both financial and non-financial value such as medical and military value.

The Need for Risk Point Metrics

Software projects are nothing if not risky. Indeed, the observed failure rate of software projects is higher than almost any other manufactured product. While software risk analysis is a maturing discipline, there are still no metrics that can indicate the magnitude of risks. Ideally, both risks and value could be analyzed together. A hypothetical value risk point metric might include the following factors:

- **Risk points**
  Risks of death or injury
  Risks to national security
  Risks of property destruction
  Risks of theft or pilferage
  Risks of litigation
  Risks of business interruption
  Risks of business slow-down
  Risks of market share loss
  Risks of schedule delays
  Risks of cost overruns
  Risks of competitive actions
  Risks of customer dissatisfaction
  Risks of staff dissatisfaction

Large software projects fail almost as often as they succeed, which is a distressing observation that has been independently confirmed.

It is interesting that project management failures in the form of optimistic estimates and poor quality control tend to be the dominant reasons for software project failures.

The bottom line is that risk analysis supported by some form of risk-point quantification might reduce the excessive number of software project failures that are endemic to the production of large software applications.

As it happens, there is extensive data available on software risks. A number of risks correlate strongly to application size measured in function points. The larger the application, the greater the number of risks will occur and the more urgent the need for risk abatement solutions.

Risk points could be combined with value points, function points, and data points for determining whether or not to fund large and complex software projects that might not succeed. While function points are useful in funding decisions, the costs of data migration and data acquisition need to be considered too, as do risk factors.

The Need for Security Points

Software and the data processed by software now control most of the major assets of the industrialized world. All citizens now have proprietary information stored in dozens of data bases: birth dates, social security number, bank account numbers; mortgages, debts, credit ratings, and dozens of other confidential topics are stored in numerous government and commercial data bases.

Hacking, worms, denial of service attacks, and identity theft are daily occurrences, and there is no sign that they will be reduced in numbers in the future.

These facts indicate a strong need for a "security point" metric that will provide quantification of the probable risks of both planned new applications and also legacy applications that process vital information.

- **Security points**
  Value of the information processed
  Volume of valuable information (using data points)
  Consequences of information theft or loss
  Consequences of disruption or denial of service
  Security flaw prevention methods
  Security attack monitoring methods
  Immediate responses for security attacks

Security as of 2015 is not as thorough as it should be. Hopefully the development of a security-point metric will encourage software developers, executives, and clients to be more proactive in avoiding security risks, and more effective in dealing with security attacks.

The purpose of security points is two fold: one is to identify in a formal manner all of the security risk topics; the second is to identify in a formal manager all of the known

security solutions. It is obvious that security cannot be fully effective by using only firewalls and external software to intercept viruses, worms, and other malware. Software needs to stronger immune system that can fight off invading malware due to better internal controls and eliminating today's practice of transferring control and exposing confidential information.

The Need for Configuration Points

This 13th metric was not developed by the author but was provided by George Stark of the IBM Global Technology Center in Austin, TX. IBM has been a pioneer in metrics research since the original function point metrics were developed at IBM White Plains in the middle 1970's.

The configuration point metric is used to predict the work effort for deploying complex suites of software and hardware that need to operate together. Unlike some of the prior metrics in this report, configuration points have existed since 2006 and have been used on a number of actual installations and seem to generate useful information.

- **Configuration Points**
  Cabling
  Software assets and configurations
  Computing assets
  Communication assets
  External interfaces

- **Value-added adjustments**
  Security
  Installation ease
  Common components
  Environment complexity
  Customizations
  External services
  Staff experience

When used for deploying large and complex combinations of software and devices, the ranges of component points to date have been between about 30,000 and 70,000. When comparing component points to standard function points, it can be seen that this metric is clearly aimed at the problems of deploying fairly massive combinations of features.

EXAMPLE OF A MULTI-METRIC ECONOMIC ANALYSIS

Because this proposed suite of metrics is hypothetical and does not actually exist as of 2015, it might be of interest to show how some of these metrics might be used. (In this small example some of the metrics aimed at large applications such as configuration points are not shown.) Let us consider an example of a small embedded device such as a smart phone or a hand-held GPS that utilizes a combination of hardware, software, and data in order to operate:

Example of Multi-Metric Economic Analysis

Development Metrics	**Number**	**Cost**	**Total**
Function points	1,000	$1,000	$1,000,000
Data points	1,500	$500	$750,000
Hardware function points	750	$2,500	$1,875,000
Subtotal	3,250	$1,115	$3,625,000
Annual Maintenance metrics			
Enhancements (micro function points)	150	$750	$112,500
Defects (micro function points)	750	$500	$375,000
Service points	5,000	$125	$625,000
Data maintenance	125	$250	$31,250
Hardware maintenance	200	$750	$150,000
Annual Subtotal	6,225	$179	$1,112,500
TOTAL COST OF OWNERSHIP (TCO)			
(Development + 5 years of usage)			
Development	3,250	$1,115	$3,625,000
Maintenance, enhancement, service	29,500	$189	$5,562,500
Data maintenance	625	$250	$156,250
Hardware maintenance	1,000	$750	$750,000
Application Total TCO	34,375	$294	$10,093,750
Risk and Value Metrics			
Risk points	2,000	$1,250	$2,500,000
Security points	1,000	$2,000	$2,000,000
Subtotal	3,000	$3,250	$4,500,000
Value points	45,000	$2,000	$90,000,000
NET VALUE	10,625	$7,521	$79,906,250
RETURN ON INVESTMENT (ROI)			$8.92

As can be seen, normal function points are used for the software portion of this product. But since it also has a hardware component and uses data, hardware points and data points are part of the cost of the application.

While smart phones are security risks, GPS devices are not usually subject to hacking in a civilian context. Therefore the risk and security totals are not high.

Value points would be based on a combination of direct revenues, indirect revenues for training and peripherals. There might also be drag-along revenues for additional services such as applications.

Note that software development itself is less than one tenth of the total cost of ownership (TCO). Note also that economic value should be based on total cost of ownership for the entire product, and not just the software component.

THE PROBABLE EFFORT AND SKILL SETS FOR CREATING A SUITE OF FUNCTIONAL METRICS

Allan Albrecht, John Gaffney, and other IBM colleagues worked on the development of function point metrics for several years before reaching a final version that achieved consistently good results.

Each of the proposed metrics in this paper would probably require a team that includes both function point experts and domain experts in topics such as data structures, hardware engineering, accounting, and other relevant topics. A single inventor might be able to derive some of these metrics, but probably a multi-disciplinary team would have more success.

Because function points already exists, creating a family of metrics that utilize similar logic would not be trivial, but would probably not be quite as difficult as the original development of function points in IBM in the 1970's. Following are the probable team sizes, skill sets, and schedules for creating a family of functional metrics:

	Metric and Skills	**Team Size**	**Schedule Months**
1.	Application function point metrics* Software engineering Accounting and finance Statistical analysis	6	24
2.	Component feature point metrics** Function point analysis Software engineering Taxonomy construction	4	12

3. Hardware function point metrics Function points Electrical engineering Mechanical engineering Aeronautical engineering Accounting and finance	6	18
4. COTS application point metrics*** Function point analysis Taxonomy construction Software engineering	1	6
5. Micro function points** Function point analysis Maintenance of software	3	3
6. Data point metrics Function point analysis Data structure analysis Data normalization methods Accounting and finance	6	18
7. Web-site point metrics Function point analysis Web site design Web content sources Graphical design Accounting and finance	6	18
8. Software usage point metrics* Function point analysis Accounting and finance	1	3
9. Service point metrics Function point analysis Info. Tech. Infrastructure. Library	4	9
10. Risk point metrics Function point analysis Software risks Software risk abatement Accounting and finance	4	6
11. Value point metrics Function point analysis	6	9

Accounting and finance Software engineering Economic modeling Multi-variate analysis		
12. Security point metrics Software security principles Costs of security breaches Function point analysis	6	6
13. Configuration points* (Developed by IBM)	NA	NA
TOTAL	53	132

* Metric currently exists
** Metric exists in prototype form
*** Metric is covered by a patent application

As can be seen, the set of possible functional metrics discussed in this paper requires substantial research. This kind of research would normally be performed either by a university or by the research division of a major company such as IBM, Microsoft, Google, Oracle, and the like. Indeed configuration points are a recent metric developed by IBM.

For example, as a data base company Oracle should certainly be interested in data point metrics and should already have data about migration costs, data quality, and the like. But as of 2015 data base and ERP installation routinely cost more than expected, while data migration efforts routinely run late and encounter data quality problems. Data economics remains a critical unknown in corporate economic studies.

The purpose of this paper is to illustrate that while function points are valuable metrics for software economic analysis, software does not exist in a vacuum and many other business and technical topics would benefit from the logic of functional metrics.

The most critical gaps in metrics as of 2015 are the lack of effective metrics for dealing with data size and data quality, and the lack of effective metrics that can integrate tangible and intangible value.

It goes without saying that the suite of metrics cannot be developed in isolation. They need to be considered as a set, and they also need to be commensurate with standard function points so that the various functional metrics can be dealt with mathematically and be used for statistical analysis as a combined set of related metrics.

The 13 proposed metrics discussed in this paper are not necessarily the only additional metrics that might useful. The fundamental point is that the function point community should expand their vision from software alone and begin to address other critical business problems that lack effective metrics and measurement techniques.

METRICS GROWTH AND CHANGE OVER MULTIPLE YEAR PERIODS

A topic that is not covered well in the metrics and function point literature is that of continuous growth and change in size. Software requirements tend to grow at rates of about 1% per month during development. After release, software applications continue to grow at about 8% per calendar year. Every few years commercial software will add "mid-life kickers" or big increases in functionality, which of course adds to function point totals.

Software Risk Master (SRM) uses a patent-pending sizing engine that predicts and accumulates size from the start of requirements through up to 10 years of post-release maintenance and enhancements.

Although this article concentrates on quality and the initial release of a software application, the Software Risk Master™ sizing algorithms actually create 15 size predictions. The initial prediction is for the nominal size at the end of requirements.
SRM also predicts requirements creep and deferred functions for the initial release. After the first release SRM predicts application growth for a 10 year period.

To illustrate the full set of SRM size predictions, the following table shows a sample application with a nominal starting size of 10,000 function points. All of the values are in round numbers to make the patterns of growth clear:

Software Risk Master™ (SRM) Multi-Year Sizing

Patent application 61434091. February 2011.

	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

As can be seen from the table software applications do not have a single fixed size, but continue to grow and change for as long as they are being used by customers or clients. Namcook Analytics and Software Risk Master (SRM) re normalize productivity and quality data on an annual basis due to changes in application size over time.

SUMMARY AND CONCLUSIONS

The value of function point metrics for economic analysis of software applications is good enough to suggest that the same logic might usefully be applied to other business topics which are difficult to measure.

The two most difficult measurement topics as of 2015 are data and value. Data lacks any metrics whatsoever, and there is no reliable information on data costs or data quality. Value has metrics for revenues and cost reduction, but no effective metrics for handling non-financial value such as medical value, military value, and many others.

REFERENCES AND READINGS

DeMarco, Tom; Why Does Software Cost So Much?; Dorset House, New York, NY; ISBN 0-9932633-34-X; 1995; 237 pages.

Fleming, Quentin W. & Koppelman, Joel M.; Earned Value Project Management; 2nd edition; Project Management Institute, NY; ISBN 10 1880410273; 2000; 212 pages.

Gack, Gary; Managing the Black Hole: The Executive's Guide to Managing Risk; The Business Expert Publisher; Thomason, GA; 2010; ISBN-10-935602-01-2.

Galorath, Daniel D. & Evans, Michael W.; Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves; Auerbach, Philadelphia, AP; ISBN 10-0849335930; 2006; 576 pages.

Garmus, David & Herron, David; Function Point Analysis; Addison Wesley, Boston, MA; ISBN 0-201069944-3; 363 pages; 2001.

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

Harris, Michael; Herron, David; Iwanicki, Stasia; The Business Value of IT; CRC Press; an Auerbach Book; 2008; ISBN13: 978-1-4200-6474-2.

Hill, Peter (editor); Practical Software Estimation; McGraw Hill, NY; 2011; ISBN 978-0-07-17191-5.

Jones, Capers; The Technical and Social History of Software Engineering, Addison Wesley, 2014.

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; Provisional Patent Application 126203 00002; January 19, 2011; A Method of Rapid Early Sizing for Software Applications; Capers Jones & Associates LLC.

Jones, Capers; A New Business Model for Function Point Metrics; Version 9.0; May 12, 2010; Capers Jones & Associates LLC.

Jones, Capers; Sizing Up Software; Scientific American Magazine; New York NY; Dec. 1998, Vol. 279 No. 6; December 1998; pp 104-109.

Jones, Capers; Applied Software Measurement; McGraw Hill, 3rd edition 2008; ISBN 978-0-07-150244-3; 575 pages; 3rd edition (March 2008).

Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA, 2000; 659 pages.

Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Version 6; Software Productivity Research, Burlington, MA; June 2006; 54 pages.

Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 2nd edition, 2007; 644 pages; ISBN13: 978- 0-07-148300-1.

Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

Kaplan, Robert S & Norton, David B.; The Balanced Scorecard; Harvard University Press, Boston, MA; ISBN 1591391342; 2004.

McConnell, Steve; Software Estimation – Demystifying the Black Art; Microsoft Press, Redmond, Wa; ISBN 10: 0-7356-0535-1; 2006.

Parthasarathy, M.A.; Practical Software Estimation – Function Point Methods for Insourced and Outsourced Projects; Addison Wesley, Boston, MA; ISBN 0-321-43910-4; 2007; 388 pages.

Stark, George; personal communication on configuration points; IBM Global Technology Center; Austin, TX; March 2006.

Strassmann, Paul; The Squandered Computer; Information Economics Press, Stamford, CT; 1997.

Stutzke, Richard D.; Estimating Software-Intensive Systems – Projects, Products, and Processes; Addison Wesley, Boston, MA; ISBN 0-301-70312-2; 2005; 917 pages.