On XML Based Automated Function Point Analysis: An Effective Method to Assess Developer Productivity

Jeffrey S. Lent and Yanzhen Qu

***Abstract*—This article proposes a novel approach to objectively measure developer productivity and individual contribution to development projects using automated function point analysis (integrated with source control system (SCS) and continuous integration (CI)). Automated function point analysis is a central tenant required in this approach. The method proposed herein relies on pattern matching to identify and group fields in to internal logical files, external inputs, external outputs, and external query objects which can be used to identify the objective size of software. Unlike earlier automated functional sizing algorithms, the method proposed is applicable to any application source code that expresses its user interface in well-formed extensible markup language (XML). This study shows that a prototype implementation of the proposed automated counting method is capable of providing a reliable and consistent relative size of existing software (accurate to +/-6.65% and 5000 times faster), and goes on to explain how this method can be integrated with existing requirements engineering tools, source control systems and continuous integration tooling in order to produce an objective measure of software developer output compared to estimates established during requirements gathering and planning that exceeds the common, subjective means that are in use in many organizations today.**

***Index Terms*—Developer productivity, function point analysis, automation, XML, functional sizing.**

I. Introduction

Function point analysis (FPA) provides an objective measure of the size of both planned and existing software [1] and provides several advantages which differentiate the process from other size analysis techniques. Because the process of generating a function point count can be rigorously applied, it is possible for different counts to be performed by different people and still obtain a result that is similar (within a margin of error of approximately 10%) [2], [3]. Finally, function point analysis has a number of advantages over other sizing techniques including consistency regardless of programming language, ability to measure non-coding activities, applicability to software reuse analysis, and providing a strong basis for software cost estimation [4]. Unfortunately function point analysis also carries with it several distinct disadvantages: the requirement of certified function point specialists in order to achieve accurate, reliable counts; they are time-consuming and expensive; automation of function point analysis has been poorly studied and is, therefore, of unknown accuracy; and the process is unreliable for software that is smaller than 15 function points in size [4]. These drawbacks have led to slow adoption of function point analysis [5].

This article presents a method of rule based function point analys is automation that can be applied to any software that expresses its user interface using XML (e.g. XAML, HTML, XUL, etc.). This method allows function point sizing to be integrated with the development teams continuous integration system so that after each successful commit the size can be measured and compared to previous measurements in order to attribute the changing size to the developer that executed the commit. This forms a basis for measuring developer productivity over time. In addition to establishing developer productivity a comparison of expected versus actual size can be calculated to control the estimation and planning process.

Generally, once requirements are established (using IEEE standard 830, or by less rigorous methods), analysts can produce a function point estimate of the expected size of the requirement. During development the product can be counted when the requirement is complete in order to establish how close to the estimated size the actual requirement turned out to be. Further, actual time spent on the requirement can be correlated with the actual size of the requirement to establish a time per function point factor that can be applied to future estimates to estimate cost. By comparing the time per function point factor established for the whole company against that of specific teams, their relative efficiency can be gauged.

While this process is an ideal method for tracking actual versus planned size (and by extension, developer effort and project cost) it is seldom used in commercial software development organizations due to the time consuming nature of manual function point analysis. The method of automated function point analysis that is presented here provides a faster method of producing function point counts that is equally accurate to the manual approach.

The method presented in this paper is focused on producing a general solution that can be used to automatically calculate the size of existing software. The solution is general because it can be used anywhere the inputs and outputs are expressed using valid XML. The solution is considered automated because most of the crucial steps can be fully executed by a computer. Certain steps are not automated, namely those that only need to be set up at the beginning of the project, such as determination of the counting boundary and determination of the type of count. Most of these non-automated steps only need to be set once per project or can rely on sensible defaults.

The work presented here seeks to present a generalized, cost effective tool that will allow objective measurement of

Manuscript received July 4, 2014; revised September 11, 2014.

The authors are with Colorado Technical University, Colorado Springs, CO 80907 USA (e-mail: Jeffery.lent@my.cs.coloradotech.edu, yqu@coloradotech.edu).

software size in order to better track programmer productivity, improve cost and duration estimation, and to allow better measurement of change as components are added or removed from the code.

II. Related Works

The ability to accurately and objectively establish the size of a software project is a fundamental metric that forms the foundation for project control, developer evaluation, product pricing and more. Allan Albrecht recognized the importance of this capability and developed a methodology for objectively sizing software which he dubbed function point analysis in his seminal 1979 paper [1]. Prior to the function point analysis the primary method for sizing software was counting lines of code (LOC). LOC analysis had several drawbacks including a need to directly inspect the source code, variability between languages and between implementation strategies. Albrecht's method, on the other hand, could be applied even before source code was written because it only relied upon examination of user facing artifacts including input and output fields and the logical structure of the data stores associated with the code. Albrecht's method gained wide acceptance in the decade following publication of his initial article.

A. Research Methodologies Used to Study FPA

Most studies that were reviewed utilized a quantitative analysis technique, which varied depending on the broad area of investigation, which was undertaken. Some studies were focused on evolving the process of function point analysis [1], [6]-[16]while other studies focused on automation of all or some of the processes involved in FPA [5], [17]-[19]. The articles, which were attempting to extend and evolve FPA generally, structured their research to compare one or more aspects of the new process they were developing against the previously established best practices for performing FPA. Most of the articles, which were working on automation, compared the results generated by their experimental process with results established by expert analysts. The analysis in this paper extends this tradition of utilizing a comparative experimental process which evaluates the differences between the manual and automated method. The analysis also draws on sampling techniques utilized for testing the efficacy of measurement tools in chemistry and other pure sciences [20].

B. Automation of Function Point Analysis

By the mid-1990s the benefits of function point analysis were becoming well accepted and well known, but serious shortcomings were seen because analysis could only be performed by human FP experts. Researchers then began to focus their efforts on investigating means to automate the analysis process. Early research by Sygma et al. addressed a method to automatically identify external inquiry points through source code evaluation, but did not provide a formal model for implementing all of the Counting Practice Manual (CPM) rules put forward by the IFPUG [17]. Although incomplete, the paper showed that it may be possible at some point to produce a fully automated method of counting existing software.

Concurrently with Sygma's work, Ho and Abran proposed a framework that can be used to build a model for automatic FP counting that would be in compliance with the IFPUG CPM. Their automated method required source code inspection of COBOL systems and produced information that could be used to assess the uniqueness of files and transactions while identifying files and transactions for the function point count. The authors experienced difficulty with establishing a concrete implementation due to the difficulty in establishing a "slicing" technique that accurately divides the source code into functional units [9]

Another approach to function point analysis utilized UML design specifications to produce FP counts. The authors produced a method which used UML version 1.1 diagrams produced by Rational Rose Version 4.0 and could be examined for particular patterns that could be translated to various IFPUG FP constructs either manually or using automated pattern matching tools. This method was an improvement over source code analysis because it could use UML diagrams that were either produced from completed source code or it could be used for predictive analysis on diagrams that are created from use case documentation prior to implementation. Class and sequence diagrams were used exclusively for the generation of FP counts [16]. A similar method of producing automated counts using COSMIC FFP methods also relied on UML diagrams [21]. Neither of these methods measures existing software.

Both of these automation methods require intimate knowledge of implementation details in order to create a function point count which, therefore, implies that internal computational details have an impact on the resulting FP counts. Albrecht's original premise implies that only inputs, outputs, queries and logical file structures are necessary to create an accurate FP count [1]. The method proposed in the next section holds true to that original premise, is compliant with the latest process detail outlined in IFPUG CPM v4.2, and can be used at any stage of the design and development process so long as the user interface and data storage schema is expressed in a machine readable format.

C. Gaps in the Body of Knowledge Related to FPA Automation

Previous studies related to automated function point analysis illustrate the current state of the art in function point analysis as a theoretical body of knowledge that is mature and well accepted within the academic community but which exists several impediments that are preventing wide spread adoption within the practitioner community. The primary impediment being the requirement that an organization must employ a highly skilled dedicated resource to manually prepare function point counts because there currently is no automated process, which can be applied, in a wide range of cases. In other words, the unit cost of producing high quality function point counts is too high to justify their use on all but the largest well-funded projects. The three primary gaps in the body of knowledge that are preventing broad adoption through automation include lack of a generalized method to apply FPA automation, the current requirement to heavily annotate software prior to the application of function point

analysis automation, and the fact that FPA can only be automated in a number of narrowly defined cases.

The most successful attempts to define an automated process have involved first establishing a method to map the countable artifacts required by the IFPUG Counting Practice Manual[22] to how those artifacts are encoded in the source code. For example, Ho & Abram propose a method of inspecting COBOL code in order to achieve such a mapping [9]. The limitation in such systems so far has been that they have required access to the source code under investigation and they have been very narrowly focused on a particular programming language. Since older languages such as COBOL and those used by other researchers in CASE systems do a poor job of separating concerns (e.g. code needed to generate the user interface was not isolated from internal business logic) it was very difficult to ascertain which parts of the code were attributable to which parts. This led Ho & Abram to consider the need for a slicing algorithm that could automatically perform such an analysis of the code. In other cases the software can only be counted if there are navigation markers embedded in the output markup which will allow the counting software to properly attribute the different function point constructs [23]. As a result of these gaps FPA automation has not become broadly accepted because they have prevented the development of a generalized tool that can be used by a wide range of software development teams. Instead only the largest projects that can afford the cost of a high degree of customization to develop the narrowly focused tools required for their specific circumstances can benefit .[22].

D. Positioning of Our Research within the Body of Knowledge

As is the case with any project, proper tools for determining the size of the project is a key foundational metric for establishing control over cost and duration. By extending the function point automation body of knowledge to fill the gaps presented above, the software development community will be one step closer to obtaining a generalized, cost effective tool that will allow for an objective measurement of this key software metric. Applications of having such a metric available in real time following each build of the software include better tracking of programmer productivity, better estimation of the cost of development of key features, the ability to measure the change in value of a product when certain components are removed or added, etc.

The approach taken in this research implements the key IFPUG rules including making clear, reasonable assumptions when establishing certain aspects of the rule set which do not lend themselves to automation (e.g. Application boundary, or differentiating external versus internal data sources). The proposed approach will allow automated counting of any application which fully exposes its user interface, and which utilizes some form of valid XML for expression of that user interface. Just as Albrecht's initial intent established the counting process will require no knowledge of the internal source code under review [1]. This approach builds upon previous research but extends it to establish a more general solution that can be applied to a broad subset of all development activities, including web application, Windows applications that utilize Microsoft's WPF technology, and XUL based applications.

III. Problem Statement and Hypothesis

Using function point analysis (FPA) allows consistent measurement of the size of a software project, but doing so manually is time consuming and labor intensive and requires staff with highly specialized knowledge of the FPA domain.

A. Problem Statement and Proposed Method

The lack of an automated and consistent method to accurately determine the size of an existing software artifact using function points has blocked FPA from being broadly accepted as an effective metric for software productivity.

The purpose of the study is to present and empirically test a method for automating function point counts for existing software products which express their user interface as well formed extensible markup (XML). The study will compare function point counts generated from a prototype implementation of the automation method against manually generated function point counts and will then produce a statistical analysis of both sets of counts to establish consistency between the two groups. In addition to establishing consistency between automated and manual counting methods, the study will also compare the time required to perform each set of analyses in hopes of showing a ten times differential between automated and manual methods.

B. Hypotheses Statement and Research Questions

It is possible, using the proposed method, to automatically (except for those attributes which can be explicitly set once per project such as determining the counting boundary, or the type of count) create a function point count of any software solution which uses well-formed XML to express its user interface which will be as accurate as counts created by experienced function point analysts.

Research Question 1: Can the proposed method produce an automated function point count that is as accurate (+/-10%) as a manual function point count?

Research Question 2: Can the proposed method perform an automated function point count faster than a manual function point count can be produced?

IV. Experiment

In order to prove the above hypothesis, a functioning prototype will be created which can be used to produce function point counts according to the rules of the IFPUG. The rule set will be adhered to whenever possible, but certain aspects of the rules will need to be approximated when the IFPUG requires the use of human judgment. Such judgments will be extracted from the prototype and expected as inputs from a human operator when possible (e.g. determination of application boundary, type of count, and value adjustment factors), or codified as a decision tree when necessary (e.g. when determining data element type (DET) inclusion in specific internal logical files (ILF), establishing whether a DET is a control or data related element, etc.).

The general method of automated analysis involves scanning the set of all XML files contained within the application boundary in order to identify all user interface elements. Application of a set of counting rules is then applied to these interface elements in order to match them to the various elements required to establish the count (e.g. DETS, ILF, EI, EO, EQ, etc.). The rules will vary between types of XML UI files but will always involve rules for mapping identified data element types (DETs) to transactional or data functions (e.g. a rule for such a mapping in XHTML would be to group all fields within a form tag together as a transactional type) and for differentiating between the various types of elements (EI vs. EO, etc.).

The prototype will perform the following tasks:

1) Collect as inputs
   - A URI that indicates the application boundary.
   - the Value Adjustment Factor
2) Scan all directories accessible from the root URI provided as input.
3) Establish a list of all DETs and assign them to ILFs following the general rule that each grouping of DETs will map to a single ILF and duplicate references should be eliminated.
4) Establish a list of external input (EI) objects and their relationships to ILFs. (using pattern matching)
5) Establish a list of external output (EO) objects and their relationships to ILFs (using pattern matching)
6) Establish a list of external query (EQ) objects and their relationships to ILFs (using pattern matching).
7) Establish the size of each ILF, EI, EO and EQ using the IFPUG rules.
8) Apply the adjustment factors provided by the operator.
9) Return the adjusted function point count.

The prototype will be designed to expect the XML files to be XAML. Later research will seek to produce a generic implementation that uses XSLT translation to convert any well-formed XML to an analysis schema so that the same code could be used to manage XHTML, XUL, Flex, or any other well-formed user interface. These XSLT translations would only need to be created once for any particular UI file format and then could be reused for all files that conform to that file format.

A. Population and Sample

The study is designed to compare the automated method against the accepted manual method of performing function point analysis on existing software artifacts. This type of study varies somewhat from studies that seek to compare an experimental group against a control group, in that the goal is to establish the variation between two methods that achieve the same end. In this study the goal is two-fold: First to establish the degree of agreement between the two measurement techniques (human vs. machine), and secondly to show the relative speed difference between the two measurement techniques. Traditional studies that seek to show correlation between different variables commonly use Pearson's product-moment correlation coefficient (*r*), however this method is not correct when comparing two methods of measuring the same variable [24]. Bland & Altman [25] establish a method of comparison such that if you consider *d* to be the mean difference and *s* to be the standard deviation of the differences, then 95% of all results would fall within +/- 2*s*. This range represents the limits of agreement between the two methods of measurement. If these bounds represent an acceptable variation then the methods can be used interchangeably. In order to establish a normal distribution of the limits of agreement it is necessary to recognize that the standard error of *d*-2*s* and *d*+2*s* is about $\sqrt{\frac{3s^2}{n}}$, where *n* is the sample size. 95% confidence intervals can be calculated by finding the appropriate point of the t distribution with n-1 degrees of freedom [25].

B. Sampling Procedure

When establishing the size and nature of the sample set for the study we look to the work of Kristian Linnet's [20] study of sample sizing guidelines for method comparison studies. Although Linnet's work was established for use in clinical chemistry the statistical basis is sound regardless of the method being studied provided there is a regulatory authority that has established an analytical tolerance which is accepted by the community. In the case of function point analysis this tolerance is +/-12% [27] which will be used to represent our critical difference (Δ). Given a measurement range of 10 FP to 140 FP, and assuming a proportional analytical standard deviation (SD_as) with an analytical coefficient of variation (CV_a) of 3% we can calculate a critical systematic difference (Δ_c) from the general formula $\Delta_c = Y_c - X_c = \alpha_0 + (\beta - 1)\, X_c$ where X_c and Y_c represent the variations in the values from the unity values and α_0 represents the variation in intercept and β represents the variation in slope of the unity function such that Δ_c = 12%-1.65 · 3% = 7%. Given a mid-point measurement of 65FP, we have a Δ_c at that level of 4.2. Therefore, we can establish that $\Delta\, \beta_{ST} = (\beta\text{-}1)/CV_a = 0.07/0.02 = 3.5$, and likewise $\Delta\, \alpha_{0ST} = (\alpha_0\text{-}0)/(CV_a \cdot X_m) = 4.2/(0.02 \cdot 65) = 3.2$ (X_m = mid-point value of the measurement range). Using Linnett's [20] "Sample size table for comparison of methods with proportional SD_as using weighted Deming regression analysis" table we see that with a range ratio of 14 and $\Delta\, \beta_{ST}$ of 3.2 we need a sample size between 10 and 12 (interpolation provides 11) so *N* should be 11. Referencing the same table we see that with a range ratio of 14 and $\Delta\, \alpha_{0ST}$ = 3.2 then we have sample size of ≤ 10 so *N* should be less than 10. Given these two options the sample size of this project should be 11 tests.

To summarize, the set of *N* samples to be counted should be no less than 11. Those samples should be evenly distributed in size so that the largest is approximately 140FP in size and the smallest is approximately 10FP in size. This sample size will ensure that the results are statistically significant (p<=0.05) with a power of at least 90%.

C. Instrumentation

The primary instrument that will be used for data collection in the study will be an excel spreadsheet that will be used to collect test results for each artifact that is tested. The Excel workbook will have a single sheet with the following column headings: Artifact Name, Manual Count Start Time, Manual Count End Time, Manual Count Duration, Automated Count Start Time, Automated Count

End Time, Automated Count Duration, Manual Function Point Size, and Automated Function Point Size.

Other instruments that will be used will be manual function point count worksheets to record the details of the work done during the manual counts as well as screen captures and log files to capture the output of the automated function point counts. Data from these secondary instruments will be used to populate the primary Excel worksheet that is discussed above. Final statistical analysis will be done with the help of SPSS, so working files for that software will also be among the secondary instruments used in the study.

D. Validity

The basis for the validity of the statistical analysis used in this study is outlined above in the population sample and sampling procedures sections and relies heavily on the standard practices put forth by Linnett [20] and Bland & Altman [25].

E. Reliability

Since the sample size for this study is relatively small (N=11) all manual function point counts will be performed by the same researcher. The automated function points will likewise, all be performed using an identical build of the prototype software and it will be run on the same hardware for all 11 samples. The input screens that will be used to operate the prototype will be the same screens that the manual counter encounters during his analysis. This approach neutralized any variation that could occur if the manual counter were to overlook a screen during the manual counting process.

F. Data Collection

Manual test results will utilize an Excel worksheet to log the details of each count, in conjunction with the primary Excel worksheet which will be used to record timing and the final results. Automated tests will emit timing and other details to the standard output channel during execution; that stream will be redirected to a text file to record the details of each run. The researcher will then transfer timing and size information from each text file to the primary Excel worksheet after the tests are run.

V. Data Analysis

The automated counting prototype was run against eleven sample programs that were readily available from open source sites like GitHub and Microsoft's Codeplex. The sizes of these subject programs were assessed using the manual counting procedures approved by the IFPUG and were found to range between 10 and 141 function points in size. The average size of the subjects was 57 points and the median size was 37 points. Each of these programs was then evaluated by the prototype. Timings of both the manual and automated methods were taken to allow for efficiency comparison. Table I shows the data that was collected for the experiment.

The variation column in table one shows that the difference between the manual and the automated count varied from 0% (the same value for both) to as much as 40% difference. The average variation in values manual and automated counts was 9.7% difference. The BookLibrary subject showed a 40% variation between manual and automated counting. This variation occurred because the BookLibrary utilizes a model-view-view model architecture that results in a number of objects to be double counted. The ability to detect and avoid double counting in these cases will be addressed in future research.

TABLE I: Raw Data Collected

Subject	Man.	Auto.	Var.	Man. Dur. (sec.)	Auto Dur. (sec.)
CurrencyConverterWPF	10	10	0.0%	128	0.06
ProductMVVM	22	22	0.0%	238	0.07
DiagramDesinger	35	29	-17.1%	356	0.11
PhotoDemo	35	35	0.0%	363	0.07
Writer	37	39	5.4%	241	0.08
ExpenseIT	39	43	10.3%	319	0.10
BookLibrary	60	84	40.0%	415	0.12
InformationManger	75	83	10.7%	1315	0.52
WPFNotepad	88	73	-17.0%	1068	0.11
Calculator	91	85	-6.6%	281	0.05
FamilyShow	141	164	16.3%	1604	0.13

The manual count, overall, counted 633 points in 6328 seconds which suggests a counting rate of approximately 10 seconds per point. The automated method counted a total of 687 points in 1.42 seconds which suggests a counting rate of approximately 2 milliseconds per point. There does not seem to be a strong correlation between the size of the program being counted and run time though. The actual runtime is dependent on the number of files in the project and the number of lines of code in each file. The automated method is approximately 5,000 times faster than manual counting, and is therefore suitable for use in an automated production continuous integration system.

VI. Conclusion and Future Works

This study is of professional significance because it increases the applicability and utility of function point analysis by removing several impediments to wider adoption of the method. By eliminating the need for a highly trained manual counter, and by improving the process of obtaining function point counts, the cost of obtaining such counts is reduced.

Cheung *et al*. [28] have shown that the variation between manual counters is 10% or less. The initial data presented in this article shows a variation of 9.7%. The conclusion drawn from this initial experiment, therefore, is that the variation between the prototype automated counting method presented here and the IFPUG manual method is approximately the same (or slightly better) and therefore this automated method can be used with the same confidence as the manual rules established by the IFPUG.

A reliable automated method for generating accurate sizes of software can be used in conjunction with a continuous integration system to measure changes in functional size with each commit from the development team. This provides the basis for a number of metrics that can be used to track team progress, developer capability and alignment with size estimates established during planning.

The results of this study represent preliminary results that

will be used to fine tune the counting algorithm in order to provide increasingly accurate results in the future. Sample 8 was an outlier in this study with a variation of 40%. Future analysis will attempt to establish the root cause of this variation in these types of programs. Currently it seems that the most accurate results from subjects that utilize forms based applications that have a clear procedural architecture. Subject 8 has an MVVM architecture that may be leading to some double counting. Future studies will also focus on how the use of the prototype counting program impacts development teams when used in a production development environment involved in more complicated system architecture such as the one described in [26].

REFERENCES

[1] A. Albrecht, "Measuring application development productivity," in *Proc. IBO Conference on Application Development*, 1979, pp. 83–92.

[2] J. Rakos, "Using function point analysis can give you sharp estimates," *Comput. Canada*, vol. 20, no. 19, p. 20, 1994.

[3] M. J. Connolley, "An empirical study of function point analysis reliability," 1990.

[4] C. Jones, "Strengths and weaknesses of software metrics," *Am. Program*, 1997.

[5] R. D. Banker, R. J. Kauffman, C. Wright, and D. Zweig, "Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment," *IEEE Trans. Softw. Eng.*, vol. 20, no. 3, pp. 169–187, Mar. 1994.

[6] A. Abran and K. Paton, "A formal notation for the rules of function point analysis," Research Report, Université duQuébec àMontréal, Software Engineering Management Laboratory, no. 247, 1995.

[7] F. Bootsma, "How to obtain accurate estimates in a real-time environment using full function points," in *Proc. 3rd IEEE Symp. Appl. Syst. Softw. Eng. Technol.*, 2000, pp. 105–112.

[8] V. Harput, H. Kaindl, S. Kramer, and D. Garching, "Extending function point analysis to object-oriented requirements," presented at 11th IEEE International Software Metrics Symposium, 2005.

[9] V. T. Ho and A. Abran, "A framework for automatic function point counting from source code," *Int. Work. Softw. Meas.*, September 8–10, 1999, pp. 248–255.

[10] D. Jeffery and G. Low, "A comparison of function point counting techniques," *Softw. Eng. IEEE*, vol. 19, no. 5, pp. 529–533, 1993.

[11] M. Jorgensen, "A systematic review of software development cost estimation studies," *Softw. Eng. IEEE*, vol. 33, no. 1, pp. 33–53, 2007.

[12] C. F. Kemerer and B. S. Porter, "Improving the reliability of function point measurement: an empirical study," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 1011–1024, 1992.

[13] H. M. Sneed and S. Huang, "Sizing maintenance tasks for web applications," in *Proc. 11th Eur. Conf. Softw. Maint. Reengineering*, 2007, pp. 171–180.

[14] S. Aspinall, "As cloud computing matures, large businesses are harnessing the benefits," *The Guardian*, 2013.

[15] C. Symons, "Function point analysis: difficulties and improvements," *Softw. Eng. IEEE Trans.*, vol. 14, no. 8718288, 1988.

[16] T. Uemura, S. Kusumoto, and K. Inoue, "Function point measurement tool for UML design specification," in *Proc. Sixth International* Software *Metrics Symposium*, 1999, pp. 62–69.

[17] A. April, E. Merlo, and A. Abran, "A reverse engineering approach to evaluate function point rules," *Reverse Eng. 1997*, pp. 1–10, 1997.

[18] T. Mukhopadhyay and S. Kekre, "Software effort models for early estimation of process control applications," *IEEE Trans. Softw. Eng.*, vol. 18, no. 10, pp. 915–924, 1992.

[19] R. Rask and P. Laamanen, "Simulation and comparison of Albrecht's function point and DeMarco's function bang metrics in a case environment," *IEEE Trans. Softw. Eng.*, vol. 19, no. 7, 1993.

[20] K. Linnet, "Necessary sample size for method comparison studies based on regression analysis," *Clin. Chem.*, vol. 45, no. 6, pp. 882–94, Jun. 1999.

[21] A. Khelifi, A. Abran, and L. Buglione, "2.4 a system of reference for software measurements with ISO 19761 (COSMIC FFP)," *Cosm. Funct. Points Theory*, vol. 19761, pp. 1–19, 2011.

[22] F. P. C. P. Ifpug, "Release 4.1," *IFPUG*, Westerville, OH, 1999.

[23] P. Fraternali, M. Tisi, and A. Bongio, "Automating function point analysis with model driven development," in *Proc. 2006 Conference of the Center for Advanced Studies on Collaborative Research*, no. 18, 2006.

[24] K. Pearson, "Contributions to the mathematical theory of evolution. II. Skew variation in homogeneous material," *Philos. Trans. R. Soc.*, London, 1895.

[25] J. M. Bland and D. G. Altman, "Statistical methods for assessing agreement between two methods of clinical measurement," *Lancet*, vol. 1, no. 8476, pp. 307–10, Feb. 1986.

[26] Y. Zhou, Y. Zhang, H. Liu, N. Xiong, and A. V. Vasilakos, "A bare-metal and asymmetric partitioning approach to client virtualization," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 40-53, 2014.

[27] P. Kampstra and C. Verhoef, *Reliability of Function Point Counts*, pp. 1–23, 2009.

[28] Y. Cheung, R. Willis, and B. Milne, "Software benchmarks using function point analysis," *Benchmarking An Int. J.*, vol. 6, no. 3, pp. 269–276, 1999.

Jeffrey S. Lent received his B.P.S in computer science and his MBA from the State University of New York and he is currently a doctoral candidate at Colorado Technical University, USA. His research interests include evidence based software engineering and the dynamics of the engineering workplace. He has 20 years of experience in software engineering and management and he is currently employed by Osmose Utilities Inc. of Atlanta, USA.

Yanzhen Qu is a professor in computer science and information technology at Colorad Technical University, USA. He received his B.Eng. in electronic engineering from Anhui University, China, M.Eng. in electrical engineering from The Science Academy of China, and Ph.D. in computer science from Concordia University, Canada. Over his 20+ years industrial career characterized by many "the world first" innovations, he has served at various enior or executive level Product R&D and IT management positions at several multinational corporations. He was also the chief system architect and the development director of several the world first very large real-time commercial software systems. At Colorado Technical University, Dr. Qu is the dissertation supervisor of over ten computer science doctoral students, and his recent research interests include cloud computing, network security, data engineering, software engineering process and methods, data mining over non-structured data, affective computing, parallel computing, artificial intelligence, scalable enterprise information management system, big data analytics as well as embedded and mobile computing. He has served as a general/program/session chair or keynote speaker in various professional conferences or workshops. He is also a visiting professor of over twenty universities. He has published many research papers in the peer reviewed conferences and professional journals, and is currently serving as a member of editorial board of several professional journals. He is a senior member of IEEE and IACSIT.