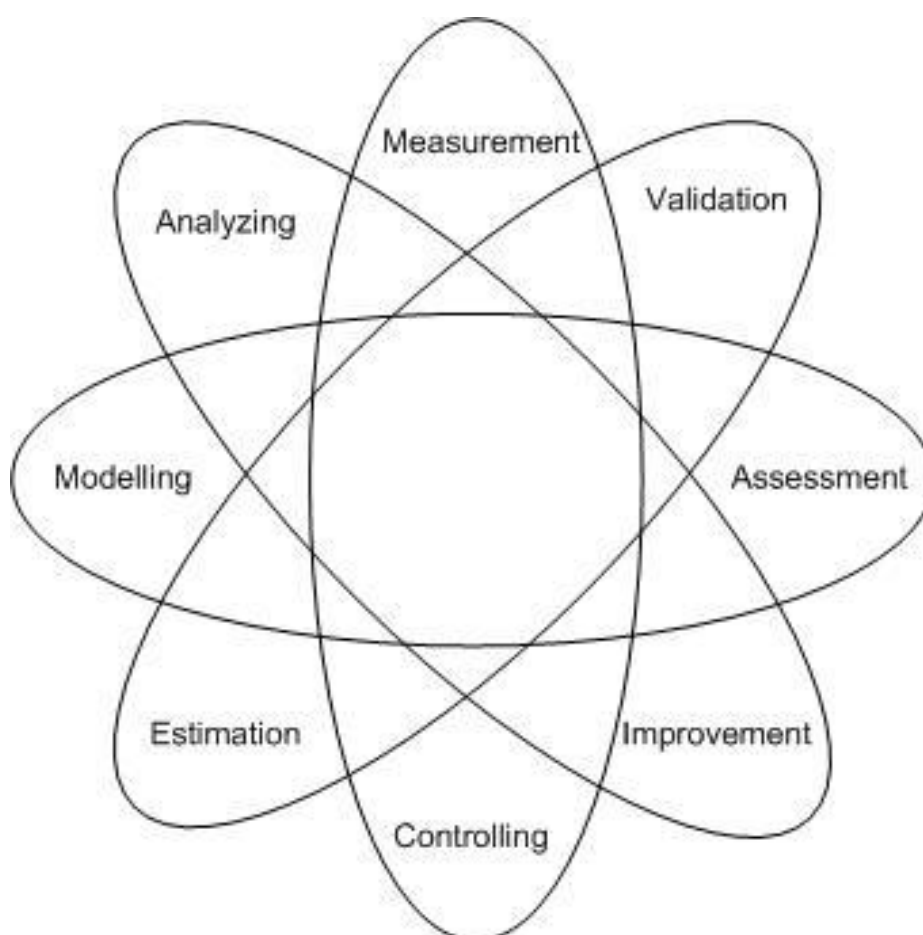


Software Measurement News

Journal of the Software Metrics Community



Editors:

Alain Abran, Manfred Seufert, Reiner Dumke, Christof Ebert, Cornelius Wille



CONTENTS

Announcements	3
Position Paper.....	9
<i>Christof Ebert</i> <i>Cyclomatic Complexity - 40 Years Later</i>	9
<i>Capers Jones</i> <i>The Origins of Function Point Metrics</i>	12
<i>Andreas Schmietendorf</i> <i>Web APIs als Enabler einer erfolgreichen Digitalisierungsstrategie</i>	15
<i>Capers Jones</i> <i>Exceeding 99% in Defect Removal Efficiency (DRE) for Software</i>	19
New Books on Software Measurement	45
Conferences Addressing Measurement Issues	49
Metrics in the World-Wide Web	51

Editors:

Alain Abran

*Professor and Director of the Research Lab. in Software Engineering Management
École de Technologie Supérieure - ETS, 1100 Notre-Dame Ouest, Montréal, Quebec, H3C 1K3,
Canada, Tel.: +1-514-396-8632, Fax: +1-514-396-8684
alain.abran@etsmtl.ca*

Manfred Seufert

*Chair of the DASMA, MediaanABS Deutschland GmbH
Franz-Rennefeld-Weg 2, D-40472 Düsseldorf, Tel.: +49 211 250 510 0
manfred.seufert@mediaan.com*

Reiner Dumke

*Professor on Software Engineering, University of Magdeburg, FIN/IKS
Postfach 4120, D-39016 Magdeburg, Germany, Tel.: +49-391-67-52812
dumke@ivs.cs.uni-magdeburg.de, <http://www.smlab.de>*

Christof Ebert

*Dr.-Ing. in Computer Science, Vector Consulting Services GmbH
Ingersheimer Str. 20, D-70499 Stuttgart, Germany, Tel.: +49-711-80670-1525
christof.ebert@vector.com*

Cornelius Wille

*Professor on Software Engineering, University of Applied Sciences Bingen
Berlinstr. 109, D-55411 Bingen am Rhein, Germany,
Tel.: +49-6721-409-257, Fax: +49-6721-409-158
wille@fh-bingen.de*

Editorial Office: Otto-von-Guericke-University of Magdeburg, FIN/IKS, Postfach 4120, 39016 Magdeburg, Germany

Technical Editor: Dagmar Dörge

The journal is published in one volume per year consisting of two numbers. All rights reserved (including those of translation into foreign languages). No part of this issues may be reproduced in any form, by photoprint, microfilm or any other means, nor transmitted or translated into a machine language, without written permission from the publisher.

© 2016 by Otto-von-Guericke-University of Magdeburg. Printed in Germany



Program

IWSM Mensura is the premier international conference on measurement and data analytics. Each year practitioners and researchers from all over the world meet to discuss practical challenges and solutions in the field of software and IT measurement and data analytics.



On **October 5-7, 2016** the IWSM Mensura conference will be held in **Berlin, Germany**. The conference venue will be at the Berlin School of Economics, Campus Lichtenberg. More information on the conference can be found on the website: <http://www.iwsm-mensura.org>.

Theme & scope

Software and IT measurement are keys for successfully managing and controlling software development projects. Data analytics and measurement are essential for both business and engineering. They enrich scientific and technical knowledge regarding both the practice of software development and empirical research in software technology. The conference focuses on all aspects of software measurement and data analytics.

This year focus is the **Value of Data**, i.e. how to maximize the value for an organization from making use of data from their software applications and systems. The trend towards digitization also dramatically increases the amount of data that becomes available. The value of a company is increasingly hidden in its data and can only be exploited fully if these are used efficiently along the entire value chain. Big data becomes an important keyword to deal with. The conference also focuses on novel approaches and innovative ideas on how to optimize existing products and processes making use of data as well as using Big Data as an enabler for new application cases.

Topics of interest

We encourage submissions in any field of software measurement, including, but not limited to:

- Practical measurement applications
- Data analytics in practice, e.g. Enterprise embedded solutions
- Usage of big data analytics for improving products and processes
- Quantitative and qualitative methods for software measurement
- Measurement processes and resources, e.g. agile or model-driven
- Empirical case studies
- System and software engineering measurement
- IT and project cost and effort estimation, e.g., cost, effort, defects
- Functional size measurement
- Data analytics and measurement in novel areas, e.g. ECU's or web services
- Measures for Cognitive Computing

Conference language

The language for the conference, workshops and special sessions is English.

CONFERENCE PROGRAM

WEDNESDAY, OCTOBER 5, 2016 (DAY 1)

Time	Track A	Titles
08:30-13:30		Conference Registration
09:00-11:30	Event 1	Benchmarking, Measurement, Data Analytics – Trends and Practice (Seminar)
10:30-12:00	Event 2	Big Data Technologies (Seminar)
13:30-14:00		Welcome and Introduction
14:00-15:00	Keynote 1	Data-Driven Innovation Study (Christian Reimsbach-Kounatze, Internet economist and policy analyst, OECD STI)
15:00-15:30		Coffee break
15:30-17:30	Session 1A	Software Evolution
15:30-16:10	1A.1	Analyzing Data on Software Evolution Processes (Harry Sneed and Wolfgang Prentner)
16:10-16:50	1A.2	Towards a Benchmark for the Maintainability Evolution of Industrial Software Systems (Till Döhmen, Magiel Bruntink, Joost Visser and Davide Ceolin)
16:50-17:30	1A.3	Evolution of Process and Product Metrics Based On Information Needs (Cenkler Yakin)
17:30-19:00	Event 4	Open COSMIC Meeting (Internal Group Meeting)
19:00-21:00		Welcome Reception

Time	Track B	Titles
09:00-12:00	Event 3	PIFs for Projects 2016 (Workshop)

15:30-17:30	Session 1B	Functional Size Measurement
15:30-16:10	1B.1	Functional Size Measurement Patterns: A Proposed Approach (Sylvie Trudel, Jean-Marc Desharnais and Jimmy Cloutier)
16:10-16:50	1B.2	Towards Component-Aware Function Point Measurement (Luigi Lavazza, Valentina Lenarduzzi and Davide Taibi)
16:50-17:30	1B.3	On the Seven Misconceptions about Functional Size Measurement (Baris Ozkan and Onur Demirörs)

Color Code

Orange	Keynote Session
Green	Co-located Event (Seminars, Workshops, or Group Meetings)
Blue	Presentation Session (Full and Short Papers, Industrial Presentations)
Grey	Break / Social Event

THURSDAY, OCTOBER 6, 2016 (DAY 2)

Time	Track A	
08:00-08:30		Conference Registration
08:30-09:30	Keynote 2	Big Data Management and Scalable Data Science (Volker Markl, Director, Berlin Big Data Center)
09:30-10:00		Coffee break
10:00-12:00	Session 2A	COSMIC
10:00-10:40	2A.1	Earned Scope Management: A Case of Study of Scope Performance using COSMIC (ISO 19761) with a Real Project (Francisco Valdes-Souto)
10:40-11:20	2A.2	An Empirical Evaluation of two COSMIC Early Estimation Methods (Luigi Lavazza and Sandro Morasca)
11:20-12:00	2A.3	Measurement of software size: Advances made by the COSMIC community (Charles Symons, Christof Ebert, Alain Abran and Frank Vogelezang)
12:00-13:00		Lunch break
13:00-14:20	Session 3A	Big Data & Analytics
13:00-13:40	3A.1	Big Data benefits for the Software Measurement Community (Jan Hentschel, Reiner Dumke and Andreas Schmietendorf)
13:40-14:00	3A.2	Quality Evaluation for Big Data: A Scalable Assessment Approach and First Evaluation Results (Michael Klaes, Wolfgang Putz and Tobias Lutz)
14:00-14:20	3A.3	Process Mining for Healthcare Process Analytics (Tugba Gurgen and Ayça Tarhan)
14:20-14:50		Coffee break
14:50-16:10	Session 4A	Process Improvement
14:50-15:30	4A.1	Risk Management: Achieving Higher Maturity & Capability Levels through the LEGO approach (Luigi Buglione, Alain Abran, Christiane Gresse von Wangenheim, Fergal McCaffery and Jean Carlo Rossa Hauck)
15:30-16:10	4A.2	Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development? (Sampo Suonsyrjä, Laura Hokkanen, Henri Terho, Kari Systä and Tommi Mikkonen)
16:10-17:10	Event 6	Data Manipulation Measurement (Workshop)
19:00-23:00		Conference Banquet

Time	Track B	
------	---------	--

10:00-12:00	Session 2B	Management
10:00-10:40	2B.1	Quality Measurement of ITIL Processes in Cloud-Systems (Anja Fiegler, Andre Zwanziger, Sebastian Herden and Reiner Dumke)
10:40-11:20	2B.2	Measurement-Based Optimization of Server License Balancing (Robert Neumann, Anja Figler, Marcus Pöhld and Reiner Dumke)
11:20-12:00	2B.3	Value of Quantitative Engagement Management - Realizing business objectives by quantitatively managing cost, time and quality dimensions of an engagement (Niteen Kumar and Cornelly Spier)

13:00-14:20	Event 5	Software Measurement in the Context of Industry 4.0 (Workshop)
-------------	---------	--

14:50-16:10	Session 4B	Metrics
14:50-15:30	4B.1	A Complexity Measure for Textual Requirements (Vard Antinyan and Mirosław Staron)
15:30-15:50	4B.2	One Metric to Combine Them All. An Experimental Comparison of Metric Aggregation Approaches (Bartosz Walter, Marcin Wolski, Patryk Promiński and Szymon Kupiński)
15:50-16:10	4B.3	Measuring the accessibility based of WCAG 2.0 Guidelines (Kathrin Wille, Cornelius Wille and Reiner Dumke)
16:10-17:10	Event 7	Estimating Packaged Software (Workshop)

FRIDAY, OCTOBER 7, 2016 (DAY 3)

Time	Track A	Titles
08:00-08:30		Conference Registration
08:30-09:30	Keynote 3	„Always On“ – Essential Capability for the Big Data Value Chain (Wolfgang Beek, CTO DACH, Software AG)
09:30-10:00		Coffee break
10:00-12:00	Session 5A	Software Quality
10:00-10:40	5A.1	A Key Performance Indicator Quality Model and Its Industrial Evaluation (Miroslaw Staron, Wilhelm Meding, Kent Niesel and Alain Abran)
10:40-11:20	5A.2	Defect Analysis in Large Scale Agile Development (Bernard Doherty, Andrew Jelfs, Aveek Dasgupta and Patrick Holden)
11:20-12:00	5A.3	Managing Large Application Portfolio with Technical Debt Related Measures (Jean-Louis Letouzey)
12:00-13:00		Lunch break
13:00-15:00	Session 6A	Estimation
13:00-13:40	6A.1	Approximation of COSMIC functional size of scenario-based requirements to support effort estimation in Agile - a replication study (Miroslaw Ochodek)
13:40-14:20	6A.2	The Missing Links in Software Estimation: Work, Team Loading and Team Power (Cigdem Gencel, Luigi Buglione)
14:20-14:40	6A.3	On Applicability of Fixed-Size Moving Windows for ANN-based Effort Estimation (Sousuke Amasaki and Chris Lokan)
14:40-15:00	6A.4	Effort Estimation in Co-located and Globally Distributed Agile Software Development: A Comparative Study (Muhammad Usman and Ricardo Britto)
15:00-15:30		Coffee break
15:30-16:30	Keynote 4	Solution-based Estimation (Eric van der Vliet, Director, CGI Global Estimation Center)
16:30-17:00		Awards and Conference Closing

10:00-12:00	Session 5B	Size Measurement
10:00-10:40	5B.1	Evaluating Security in Web Application Designs Using Functional and Structural Size Measurements (Hela Hakim, Asma Sellami and Hanene Ben Abdallah)
10:40-11:00	5B.2	Towards semi-automatic size measurement of user interfaces in web applications with IFPUG SNAP (Hassan Mansoor and Miroslaw Ochodek)
11:00-11:20	5B.3	A Proposal on Requirements for COSMIC FSM Automation from Source Code (Ayça Tarhan, Barış Özkan and Gonca Canan İçöz)

13:00-15:00	Event 8	Metrics in Contracts (Workshop)
-------------	---------	---------------------------------



BSAO/BCloud 2016

(Qualitative und quantitative Bewertung)

03.11.2016, Gastgeber Zalando, Berlin

Der diesjährige BSOA/BCloud-Workshop findet am 03.11.2016 in Berlin (Gastgeber Zalando) statt. Im Mittelpunkt der Vorträge, Diskussionsrunden und des World Cafes stehen domänenspezifische und wirtschaftliche Bewertungsfragen von Service APIs. Im Einzelnen geht es um die Identifikation, Gestaltung, Bewertung sowie das Management von Service APIs im Diskurs verschiedener Branchen (z.B. Banken, Versicherungen, Pharmazie) auseinander.

Beispiele für Themenbereiche:

- Welchen Einfluss haben Service APIs auf die Industrialisierung unternehmerische Prozessabläufe?
- Bewertung der mit Service APIs einhergehenden Möglichkeiten, im Sinne innovativer Produkte und Dienstleistungen?
- Bewertungsansätze im Zusammenhang mit der Identifikation, Spezifikation, Bewertung und Qualitätssicherung von Serviceangeboten.
- Gestaltung von Architekturen zur serviceorientierten Verzahnung von unternehmensinternen Lösungen mit Service APIs.
- Herausforderungen der Serviceorientierung im Kontext eines kollaborativen und interoperablen IT-Service-Managements.
- Gewährleistung von Sicherheits- und Compliance-Aspekten in interoperablen Architekturansätzen.

Ein besonderes Highlight erwartet die Teilnehmer mit dem eingeladenen Keynote-Sprecher Herrn *Michael Binzen* (Chefarchitekt DB Systel GmbH).

Web-Adresse zum Workshop:

<http://www-ivs.cs.uni-magdeburg.de/~gi-bsoa/2016/>

Cyclomatic Complexity - 40 Years Later

Christof Ebert
Vector Consulting Services, Stuttgart
September 2016

The criticality and risk of software is defined by its complexity. Forty years ago, McCabe introduced his famous cyclomatic complexity (CC) metric. Today, it is still one of the most popular and meaningful measurements for analyzing code. Read this blog about the measurement and its value for improving code quality and maintainability...

It is of great benefit for projects to be able to predict software components likely to have a high defect rate or which might be difficult to test and maintain. It is of even more value having an indicator which can provide constructive guidance on how to improve the quality of code. This is what the cyclomatic complexity (CC) metric gives us.

The CC metric is simple to calculate and intuitive to understand. It can be taught quickly. Control flows in code are analyzed by counting the decisions, i.e., the number of linear independent paths through the code under scrutiny. Too many nested decisions make the code more difficult to understand due to the many potential flows and possibilities of passing through it.

In addition, the CC value of a module correlates directly with the number of test cases necessary for path coverage, so even a rough indication given by the CC metric is of high value to a developer or project manager.

A high CC thus implies high criticality and the code will have a higher defect density (vis-à-vis code with a relatively lower CC); test effort is higher and maintainability severely reduced. These relationships are intuitive for students as well as experts and managers and this is another appealing feature of the CC metric.

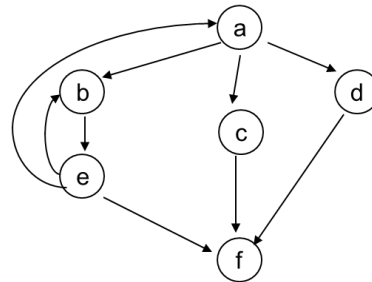
It is small wonder therefore that CC, unlike many other metrics which have been proposed over the past decades is still going strong and is used in almost all tools for criticality prediction and static code analysis.

Source code

```

a:  switch ...
    case b
b:      ...
        do e while ...
c:      ...
    case d
d:      ...
e:  if ...
    jump a
    case c
f:  end

```



Cyclomatic complexity

$$\begin{aligned}
 v(G) &= e - n + 2p \\
 &= 9 - 6 + 2 = 5
 \end{aligned}$$

= Number of different sections of the control flow graph
 = Number of binary decisions + 1

Figure: Calculation of Cyclomatic Complexity by counting linear independent paths through a control flow.

CC, together with change history, past defects and a selection of design metrics (e.g., level of uninitialized data, method overriding and God classes) can be used to build a prediction model. Based on a ranked list of module criticality used in a build, different mechanisms namely refactoring, re-design, thorough static analysis and unit testing with different coverage schemes can then be applied. The CC metric therefore gives us a starting point for remedial maintenance effort.

Instead of predicting the number of defects or changes (i.e., algorithmic relationships) we consider assignments to classes (e.g., “defect-prone”). While the first goal can be achieved more or less successfully with regression models or neural networks mainly in finished projects, the latter goal seems to be adequate for predicting potential outliers in running projects, where precision is too expensive and not really necessary for decision support. Christof – I am not sure I follow the point being made in these last two sentences – can you possibly clarify/elaborate please?

While the benefits of CC are clear, it does need clear counting rules. These days for instance, we do not count simple “switch” or “case” statements as multiplicities of “if, then, else” decisions. Moreover, the initial proposal to limit CC to seven plus/minus two per entity is no longer taken as a hard rule, because boundaries for defect-prone components are rather fuzzy and multi-factorial.

Having identified such overly critical modules, risk management must be applied. The most critical and most complex of the analyzed modules, for instance, the top 5, are candidates for redesign. For cost reasons mitigation is not only achieved with redesign. The top 20% should have a thorough static code analysis, and the top 80% should be at least unit tested with C0 coverage of 100%. By concentrating on these critical components the productivity of quality assurance is increased.

Critical modules should at least undergo a flash review and subsequent refactoring, redesign or rewriting – depending on their complexity, age and reuse in other projects. Refactoring includes reducing size, improving modularity, balancing cohesion and coupling, and so on. For instance, apply thorough unit testing with 100 percent C0 coverage (statement coverage) to those modules ranked most critical. Investigate the details of the selected modules’ complexity measurements to determine the redesign approach. Typically, the different complexity measurements will indicate the approach to follow. Static control flow analysis tools incorporating CC can also find security vulnerabilities such as dead code, often used as backdoors for hijacking software.

Our own data but also many published empirical studies demonstrate that a high decision-to-decision path coverage or C1 coverage will find over 50% of defects, thus yielding a strong business case in favor of using CC. On the basis of the results from many of our client projects and taking a conservative ratio of only 40 percent defects in critical components, criticality prediction can yield at least a 20 percent cost reduction for defect correction.

The additional costs for the criticality analysis and corrections are in the range of few person days per module. The necessary tools such as Coverity, Klocwork, Lattix, Structure 101, SonarX, SourceMeter, are off the shelf and account for even less per project. These criticality analyses provide numerous other benefits, such as the removal of specific code-related risks and defects that otherwise are hard to identify (for example, security flaws).

CC clearly has its value for critically predictions and thus improving code quality and reducing technical debt. Four decades of validity and usage is a tremendous time in software, and I congratulate McCabe for such a ground-breaking contribution.

Literature and media:

McCabe, T.J.: A Complexity Measure. IEEE Transactions on Software Engineering, Vol. SE-2, NO.4, Dec. 1976.
<http://www.literateprogramming.com/mccabe.pdf>

Selected white papers on quality practices from our media-center:
http://consulting.vector.com/vc_download_en.html?product=quality

Full article on static code analysis technologies in IEEE Software:
<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4814967>

Author:

Christof Ebert is the managing director of Vector Consulting Services. He is on the IEEE Software editorial board and teaches at the University of Stuttgart and the Sorbonne in Paris.

Contact him at christof.ebert@vector.com



The Origins of Function Point Metrics

Capers Jones

(IFPUG publication 05/17/2016, permitted by the author)

Version 3.0

VP and CTO, Namcook Analytics LLC

Email: Capers.Jones3@gmail.com

Introduction

The author was working at IBM in the 1960's and 1970's and was able to observe the origins of several IBM technologies such as inspections, parametric estimation tools, and function point metrics. This short paper discusses the origins and evolution of function point metrics.

In the 1960's and 1970's IBM was developing new programming languages such as APL, PL/I, PL/S etc. IBM executives wanted to attract customers to these new languages by showing clients higher productivity rates.

As it happens the compilers for various languages were identical in scope and had the same features. Some older compilers were coded in assembly language while newer compilers were coded in PL/S, which was a new IBM language for systems software.

When we measured the productivity of assembly-language compilers versus PL/S compilers using "lines of code" (LOC) we found that even though PL/S took less effort, the LOC metric of LOC per month favored assembly language.

This problem is easiest to see when comparing products that are almost identical but merely coded in different languages. Compilers, of course, are very similar. Other products besides compilers that are close enough in feature sets to have their productivity negatively impacted by LOC metrics are PBX switches, ATM banking controls, insurance claims handling, and sorts.

To show the value of higher-level languages the first IBM approach was to convert high-level languages into "equivalent assembly language." In other words we measured productivity against a synthetic size based on assembly language instead of against true LOC size in the actual higher level languages. This method was used by IBM from around 1968 through 1972.

An IBM vice president, Ted Climis, said that IBM was investing a lot of money into new and better programming languages. Neither he nor clients could understand why we had to use the old assembly language as the metric to show productivity gains for new languages. This was counter-productive to the IBM strategy of moving customers to better programming languages. He wanted a better metric that was language independent and could be used to show the value of all IBM high-level languages.

This led to the IBM investment in function point metrics and to the creation of a function-point development team under Al Albrecht at IBM White Plains. Function Point metrics were developed by the IBM team by around 1975 and used internally and successfully. In 1978 IBM placed function point metrics in the public domain and announced them via a technical paper given by Al Albrecht at a joint IBM/SHARE/Guide conference in Monterey, California.

Table 1 shows the underlying reason for the IBM function point invention based on the early comparison of assembly language and PL/S for IBM compilers.

Table 1 shows productivity in four separate flavors:

1. Actual lines of code in the true languages.
2. Productivity based on "equivalent assembly code."
3. Productivity based on "function points per month."
4. Productivity based on "work hours per function point."

Note: table 1 uses simple round numbers to clarify the issues noted with LOC metrics.

Table 1: IBM Function Point Evolution Circa 1968-1975

(Results for two IBM compilers)

	Assembly Language	PL/S Language
Lines of code (LOC)	17,500.00	5,000.00
Months of effort	30.00	12.50
Hours of effort	3,960.00	1,650.00
LOC per month	583.33	400.00
Equivalent assembly	17,500.00	17,500.00
Equiv. Assembly MO	583.33	1,400.00
Function points	100.00	100.00
Function Points/month	3.33	8.00
Work hours per FP	39.60	16.50

The three rows highlighted in blue show the crux of the issue. LOC metrics tend to penalize high-level languages and make low-level languages such as assembly look better than they really are. Function points metrics, on the other hand, show tangible benefits from higher-level programming languages and this matches the actual expenditure of effort and standard economic analysis. Productivity of course is defined as ***“goods or services produced per unit of labor or expense.”*** The creation and evolution of function point metrics was based on a need to show IBM clients the value of IBM’s emerging family of high-level programming languages such as PL/I, APL, and others. This is still a valuable use of function points since there are more than 3,000 programming languages in 2016 and new languages are being created at a rate of more than one per month. Another advantage of function point metrics vis a vis LOC metrics is that function points can measure the productivity of non-coding tasks such as creation of requirements and design documents. In fact function points can measure all software activities, while LOC can only measure coding. Up until the explosion of higher-level programming languages occurred, assembly language was the only language used for systems software (the author programmed in assembly for several years when starting out as a young programmer).

With only one programming language LOC metrics worked reasonably well. It was only when higher-level programming languages appeared that the LOC problems became apparent. It was soon realized that the essential problem with the LOC metric is really nothing more than a basic issue of manufacturing economics that had been understood by other industries for over 200 years.

This is a fundamental law of manufacturing economics: ***“When a manufacturing process has a high percentage of fixed costs and there is a decline in the number of units produced, the cost per unit will go up.”***

The software non-coding work of requirements, design, and documentation act like fixed costs. When there is a move from a low-level language such as assembly to a higher-level language such as PL/S, the cost per unit will

go up, assuming that LOC is the “unit” selected for measuring the product. This is because of the fixed costs of the non-code work and the reduction of code “units” for higher-level programming languages. Function point metrics are not based on code at all, but are an abstract metric that defines the essence of the features that the software provides to users. This means that applications with the same feature sets will be the same size in terms of function points no matter what languages they are coded in. Productivity and quality can go up and down, of course, but they change in response to team skills. Once function points were released by IBM in 1978 other companies began to use them, and soon the International Function Point User’s Group (IFPUG) was formed in Canada. Today in 2016 there are hundreds of thousands of function point users and hundreds of thousands of benchmarks based on function points. There are also several other varieties of function points such as COSMIC, FISMA, NESMA, etc.

Overall function points have proven to be a successful metric and are now widely used for productivity studies, quality studies, and economic analysis of software trends. Function point metrics are supported by parametric estimation tools and also by benchmark studies. There are also several flavors of automatic function point tools. There are also function point associations in most industrialized countries. There are also ISO standards for functional size measurement. (There was never an ISO standard for code counting and counting methods vary widely from company to company and project to project. In a benchmark study performed for a “LOC” shop we found four sets of counting rules for LOC that varied by over 500%). Table 2 shows countries with increasing function point usage circa 2016, and it also shows the countries where function point metrics are now required for government software projects.

Table 2: Countries Expanding Use of Function Points 2016

1	Argentina	
2	Australia	
3	Belgium	
4	Brazil	Required for government contracts 2008
5	Canada	
6	China	
7	Finland	
8	France	
9	Germany	
10	India	
11	Italy	Required for government contracts
12	Japan	Required for government contracts
13	Malaysia	Required for government contracts
14	Mexico	
15	Norway	
16	Peru	
17	Poland	
18	Singapore	
19	South Korea	Required for government contracts
20	Spain	
21	Switzerland	
22	Taiwan	
23	The Netherlands	
24	United Kingdom	
25	United States	

Several other countries will probably also mandate function points for government software contracts by 2017. Eventually most countries will do this. In retrospect function point metrics have proven to be a powerful tool for software economic and quality analysis.

Web APIs als Enabler einer erfolgreichen Digitalisierungsstrategie

Andreas Schmietendorf

Hochschule für Wirtschaft und Recht Berlin
Email: andreas.schmietendorf@hwr-berlin.de

1. Motivation

Klassische Unternehmen, wie z.B. in der Automobil- und Maschinenbaubranche, Banken, Versicherungen, Versorger oder Speditionen, waren durch eine massive Ressourcenbindung (z.B. Rohstoffe, Anlagen, Fuhrpark, Personal) und Fertigungstiefe gekennzeichnet. Für die Wettbewerbsfähigkeit moderner Unternehmen spielen aktuelle und konsistente Kenntnisse der Kundenbedürfnisse, die Innovationsfähigkeit, die bedarfsgerechte und agile Akquise von Ressourcen sowie vor allem die Möglichkeiten zur Abdeckung von globalen Märkten eine entscheidende Rolle. Aufgrund der Omnipräsenz von Software können diese Einflüsse bzw. Anforderungen nur über einfach integrierbare IT-Lösungen, die an den Unternehmensgrenzen keinen Halt machen beherrscht werden. Web-APIs können entsprechend [Spencer 2015] das strategische, fachliche und technologische Rückrad dieser unternehmensübergreifend wirkenden Integrationsanforderungen bilden.

„Application Programming Interfaces (API's) have gone from a something that only developers and architects once discussed to emerge as a capability that is central to many successful companies business strategies and a key focus of many of their senior leadership teams.“

Werden Web-APIs im Sinne eines zusätzlichen Vertriebskanals für Drittanbieter bereitgestellt, wird häufig auch von einer API economy gesprochen. Neben der ökonomischen Perspektive sieht [Tang 2015] darin ein Gestaltungsprinzip für kompositorisch orientierte Softwarearchitekturen, welches die Möglichkeiten moderner Web-APIs mit korrespondierenden Geschäftsmodellen kombiniert. Ohne einen Anspruch auf Vollständigkeit zu erheben, finden sich die Ursachen in den folgenden Aspekten:

- Web-APIs als Rückgrad mobiler Applikationen.
- Web-APIs als „Enabler“ im Diskurs des IoT.
- Web-APIs als zusätzlicher Vertriebskanal.
- Web-APIs als Datenquelle für Big Data.
- Web-APIs als Kollaborationsplattform für soziale Medien.

Neben den primär wirtschaftlich und fachlich geprägten Einflüssen existieren auch technologische Treiber, wie z.B. das Cloud-Computing, Agilitätsanforderungen im Software-Engineering oder aber die konkret eingesetzte Schnittstellentechnologie. Diesbezüglich findet sich der Einsatz von RESTful-, XML/SOAP-, JSON- oder auch programmiersprachspezifische Web-APIs, welche zumeist HTTP als Übertragungsprotokoll im Internet benutzen.

2. Digitalisierung – Industrialisierung der IT

Moderne Unternehmen mit einer agilen Sourcingstrategie profitieren von den Möglichkeiten einer umfänglichen Digitalisierung, da die für das Geschäft benötigten Daten, Funktionen und Algorithmen über fachlich spezialisierte Service APIs aus dem Internet „ad hoc“ bezogen werden. Damit wird die unternehmerische IT selbst zum Gegenstand der Industrialisierung. Für den Fall, dass diese nicht als Kernkompetenz wahrgenommen wird, kommt es zu einer dramatischen Reduktion der Fertigungstiefe im gesamten Lebenszyklus benötigter Softwarelösungen. Damit einher gehen Konsolidierungen der betroffenen

Prozesse und Organisationen. Aus diesen resultieren veränderte Kompetenzbedürfnisse, aber auch soziologische und gesellschaftliche Implikationen.

Im Diskurs der Digitalisierung stellt sich für alle Unternehmen die Frage, welchen Wertbeitrag unternehmensinterne Daten darstellen und ob diese via Web APIs (Online) oder auch als Dateien (Offline) im Internet zur Verfügung gestellt werden sollten. Die Bereitstellung unternehmensintern akquirierter Informationen via Service APIs wird aktuell zumeist als ein Risiko, denn als Change zur Bewältigung der Herausforderungen einer zunehmend digitalisierten Welt bewertet. In Abhängigkeit der aktuellen Marktpresenz können Innovationen so kurzzeitig behindert bzw. zurück gehalten werden. Allerdings entsagt sich das betroffene Unternehmen so auch der Möglichkeiten von entsprechenden Interessengruppen und Partnerschaften zu profitieren, einen zwingend benötigten Lernprozess in Gang zu setzen und nicht zuletzt die Wünsche der Kunden analytisch bewerten und damit aktiv mit gestalten zu können. Ein derartiges Umfeld birgt die Gefahr, sich vom digitalen Fortschritt abzukoppeln.

Es gilt zu klären, inwieweit die Innovations- und Wettbewerbsfähigkeit der Unternehmen unter dieser „Abschottungspolitik“ leidet, da der kreative Umgang mit existierenden Informationen an den Unternehmensgrenzen halt macht.

„Innovationen entstehen erst durch Assoziationen und das Übersetzen von Vorhandenem in neue Kontexte.“¹

Um das mit der Digitalisierung einhergehende Potential für den deutschen bzw. europäischen Standort wirtschaftlich nutzen zu können, bedarf es regulatorischer Maßnahmen von Seiten des Gesetzgebers. Nur so kann für kleinere und junge Unternehmen der Zugang zum „Rohstoff des 21. Jahrhunderts – den Daten“ gewährleistet werden, so dass kreative Lösungsansätze nicht an der Behäbigkeit und Geschlossenheit marktbeherrschender Unternehmen und ihrer Lobbyisten scheitern. Noch haben singular betrachtet Web APIs einen geringen Einfluss auf existierende Unternehmensprozesse. Die globale API Economy besitzt allerdings das Potential, virtualisierte Wertschöpfungsketten agil zu etablieren und damit unternehmerische Aktivitäten zu revolutionieren. Bei immer kürzer werdenden Innovationszyklen und Produkten, die über Software definiert werden, wird die Geschwindigkeit, mit der Lösungen am Markt platziert werden können, zum entscheidenden Wettbewerbsfaktor.

3. Qualitative Anforderungen an Web-APIs

Wer von angebotenen Web-APIs profitieren möchte und eine Einbindung in die eigenen Geschäftsprozesse vorsieht, um diese mit Informationen und Funktionen anzureichern bzw. zu optimieren, der muss von Anfang an großen Wert auf die Service-Qualität legen [Schmietendorf 2016].

Es gilt, Web-APIs langfristig und stabil in bestehende Strukturen integrieren zu können, ohne dass im Zweifelsfall schwerwiegende Konsequenzen drohen. Gleichzeitig muss es für den Entwickler möglich sein, den externen Service mit geringem Aufwand einzubinden. Da die hinter einer Web-API liegenden Implementierungen im Sinne einer Black-Box zumeist verborgen bleiben, bedarf es für die Integration einer einfachen, sicheren, aber dennoch komfortabel zu handhabenden Schnittstelle [apigee 2012]. Die in Anlehnung an [Musser 2014] erstellte Grafik zeigt ausgewählte Problembereiche von Web-APIs und mögliche Ansätze zur Lösung.

¹ Quelle: Thomas Sattelberger: Wir brauchen Biotope für die Entwicklung von Neuem
<http://goodimpact.org>, 31. März 2016

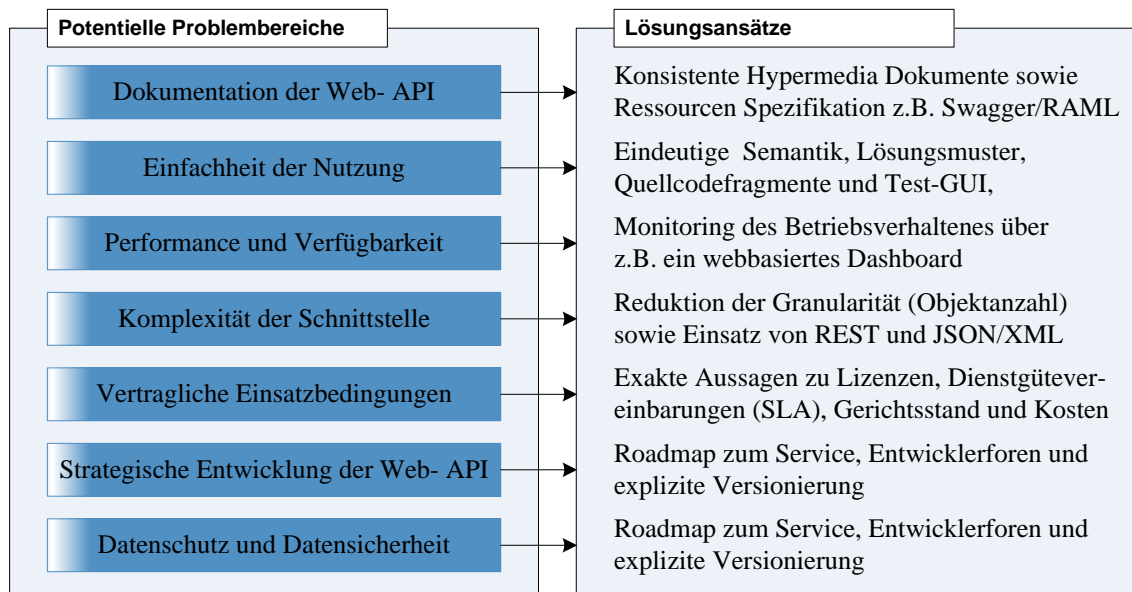


Abbildung 1: Qualitätsaspekte einer Web-API [Schmietendorf 2016]

Die zunehmende Bedeutung von Web-APIs im Bereich zur Verfügung gestellter Methoden des maschinellen Lernens (Machine Learning) oder auch der natürlich sprachlichen Programmierung (Natural Language Programing) impliziert ein notwendiges Vertrauen in die Richtigkeit der verwendeten Algorithmen. Entsprechende Beispiele finden sich mit IBM Bluemix² und den Watson Service APIs, dem Azure ML Studio³ von Microsoft oder auch dem Marktplatz Algorithmia⁴.

In diesem Zusammenhang empfiehlt sich eine Plausibilisierung mit Hilfe von Zertifikaten, die durch „vertrauenswürdige Dritte“ bereitgestellt werden. Ggf. bietet sich auch eine quelloffene Implementierung der Web-API selbst an. Je nach Art der verwendeten Open-Source-Lizenzen sind dabei Risiken in Bezug auf Compliance-Anforderungen zu prüfen.

4. Veranstaltungshinweis

Abschließend sei noch auf den diesjährigen BSOA/BCloud-Workshop am 03.11.2016 in Berlin (Gastgeber Zalando) verwiesen. Im Mittelpunkt der Vorträge, Diskussionsrunden und des World Cafes stehen domänenspezifische und wirtschaftliche Bewertungsfragen von Service APIs. Im Einzelnen geht es um die Identifikation, Gestaltung, Bewertung sowie das Management von Service APIs im Diskurs verschiedener Branchen (z.B. Banken, Versicherungen, Pharmazie) auseinander.

Beispiele für Themenbereiche:

- Welchen Einfluss haben Service APIs auf die Industrialisierung unternehmerische Prozessabläufe?
- Bewertung der mit Service APIs einhergehenden Möglichkeiten, im Sinne innovativer Produkte und Dienstleistungen?
- Bewertungsansätze im Zusammenhang mit der Identifikation, Spezifikation, Bewertung und Qualitätssicherung von Serviceangeboten.

² <https://console.ng.bluemix.net/catalog/>

³ <https://studio.azureml.net/>

⁴ <https://algorithmia.com>

- Gestaltung von Architekturen zur serviceorientierten Verzahnung von unternehmensinternen Lösungen mit Service APIs.
- Herausforderungen der Serviceorientierung im Kontext eines kollaborativen und interoperablen IT-Service-Managements.
- Gewährleistung von Sicherheits- und Compliance-Aspekten in interoperablen Architekturansätzen.

Ein besonderes Highlight erwartet die Teilnehmer mit dem eingeladenen Keynote-Sprecher Herrn *Michael Binzen* (Chefarchitekt DB Systel GmbH).

Web-Adresse zum Workshop:

<http://www-ivs.cs.uni-magdeburg.de/~gi-bsoa/2016/>

5. Quellenverzeichnis

- [apigee 2012] apigee, "Web API Design," apigee, 1 March 2012, [Online]. Available: <https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>. [Accessed 2 July 2016].
- [Musser 2014] Musser, J.: Ten Reasons Developers Hate Your API (and what to do about it), GlueCon 2014. [Online]. Available: <http://www.programmableweb.com/news/10-reasons-why-developers-hate-your-api/2014/05/23>. [Accessed 4 July 2016].
- [Schmietendorf 2016] Schmietendorf, A.; Nadobny, K.; Hentschel, J.: Design Guidelines zur konstruktiven Qualitätssicherung von Web-APIs, SQ Magazin: Ausgabe 40, ASQF, S. 18-19, September 2016
- [Spencer 2015] Spencer, S.: The Service Oriented Business and how API's power the Service Oriented Startup, APIdays Sydney/Australia, February 2015, [Online]. Available: http://syd.apidays.io/APIdays_program.pdf [Accessed 2 July 2016].
- [Tang 2014] Tang, L.: API Governance and Management, Service Technology Magazine, September/October 2014

Exceeding 99% in Defect Removal Efficiency (DRE) for Software

Capers Jones

Draft 12.0 September 8, 2016
VP and CTO, Namcook Analytics LLC



Abstract

Software quality depends upon two important variables. The first variable is that of “*defect potentials*” or the sum total of bugs likely to occur in requirements, architecture, design, code, documents, and “bad fixes” or new bugs in bug repairs. Defect potentials are measured using function point metrics, since “lines of code” cannot deal with requirements and design defects. (This paper uses IFPUG function points version 4.3. The newer SNAP metrics are only shown experimentally due to insufficient empirical quality data with SNAP as of 2016. However an experimental tool is included for calculating SNAP defects.) The second important measure is “*defect removal efficiency (DRE)*” or the percentage of bugs found and eliminated before release of software to clients. The metrics of *Defect Potentials* and *Defect Removal Efficiency (DRE)* were developed by IBM circa 1973 and are widely used by technology companies and also by insurance companies, banks, and other companies with large software organizations. The author’s Software Risk Master (SRM) estimating tool predicts defect potentials and defect removal efficiency (DRE) as standard quality outputs for all software projects.

Web: www.Namcook.com

Email: Capers.Jones3@gmail.com

Introduction

Defect potentials and defect removal efficiency (DRE) are useful quality metrics developed by IBM circa 1973 and widely used by technology companies as well as by banks, insurance companies, and other organizations with large software staffs.

This combination of defect potentials using function points and defect removal efficiency (DRE) are the only accurate and effective measures for software quality. The “Cost per defect metric” penalizes quality and makes buggy software look better than high-quality software. The “Lines of code (LOC)” metric penalizes modern high-level languages. The LOC metric can’t measure or predict bugs in requirements and design. The new technical debt metric only covers about 17% of the true costs of poor quality.

Knowledge of effective software quality control has major economic importance because for over 50 years the #1 cost driver for the software industry has been the costs of finding and fixing bugs. Table 1 shows the 15 major cost drivers for software projects in 2016. The cost drivers highlighted in red are attributable to poor software quality:

Table 1: U.S. Software Costs in Rank Order:

- 1) The cost of finding and fixing bugs**
- 2) The cost of cancelled projects**
- 3) The cost of producing English words
- 4) The cost of programming or code development
- 5) The cost of requirements changes
- 6) The cost of successful cyber-attacks**
- 7) The cost of customer support
- 8) The cost of meetings and communication
- 9) The cost of project management
- 10) The cost of renovation and migration
- 11) The cost of innovation and new kinds of software
- 12) The cost of litigation for failures and disasters**
- 13) The cost of training and learning
- 14) The cost of avoiding security flaws
- 15) The cost of assembling reusable components

Table 1 illustrates an important but poorly understood economic fact about the software industry. Four of the 15 major cost drivers can be attributed specifically to poor quality. The poor quality of software is a professional embarrassment and a major drag on the economy of the software industry and for that matter a drag on the entire U.S. and global economies. Poor quality is also a key reason for cost driver #2. A common reason for cancelled software projects is because quality is so bad that schedule slippage and cost overruns turned the project return on investment (ROI) from positive to negative. Note the alarming location of successful cyber-attacks in 6th place (and rising) on the cost-driver list. Since security flaws are another form of poor quality it is obvious that high quality is needed to deter successful cyber-attacks. Poor quality is also a key factor in cost driver #12 or litigation for breach of contract. (The author has worked as an expert witness in 15 lawsuits. Poor software quality is an endemic problem with breach of contract litigation. In one case against a major ERP company, the litigation was filed by the company's own shareholders who asserted that the ERP package quality was so bad that it was lowering stock values!) A chronic weakness of the software industry for over 50 years has been poor measurement practices and bad metrics for both quality and productivity. For example many companies don't even start quality measures until late testing, so early bugs found by inspections, static analysis, desk checking, and unit testing are unmeasured and invisible.

If you can't measure a problem then you can't fix the problem either. Software quality has been essentially unmeasured and therefore unfixed for 50 years. This paper shows how quality can be measured with high precision, and also how quality levels can be improved by raising defect removal efficiency (DRE) up above 99%, which is where it should be for all critical software projects. Software defect potentials are the sum total of bugs found in requirements, architecture, design, code, and other sources of error. The approximate U.S. average for defect potentials is shown in table 2 using IFPUG function points version 4.3:.

Table 2: Average Software Defect Potentials circa 2016 for the United States

- | | |
|-----------------------|--|
| • Requirements | 0.70 defects per function point |
| • Architecture | 0.10 defects per function point |
| • Design | 0.95 defects per function point |
| • Code | 1.15 defects per function point |
| • Security code flaws | 0.25 defects per function point |
| • Documents | 0.45 defects per function point |
| • Bad fixes | 0.65 defects per function point |
| • Totals | 4.25 defects per function point |

Note that the phrase “bad fix” refers to new bugs accidentally introduced in bug repairs for older bugs. The current U.S. average for bad-fix injections is about 7%; i.e. 7% of all bug repairs contain new bugs. For modules that are high in cyclomatic complexity and for “error prone modules” bad fix injections can top 75%. For applications with low cyclomatic complexity bad fixes can drop below 0.5%. Defect potentials are of necessity measured using function point metrics. The older “lines of code” metric cannot show requirements, architecture, and design defects not any other defect outside the code itself. (As of 2016 function points are the most widely used software metric in the world. There are more benchmarks using function point metrics than all other metrics put together.) Because of the effectiveness of function point measures compared to older LOC measures an increasing number of national governments are now mandating function point metrics for all software contracts. The governments of Brazil, Italy, Japan, Malaysia and South Korea now require function points for government software. Table 3 shows the countries with rapid expansions in function point use:

Table 3 Countries Expanding Use of Function Points 2016

1	Argentina	
2	Australia	
3	Belgium	
4	Brazil	Required for government contracts
5	Canada	
6	China	
7	Finland	
8	France	
9	Germany	
10	India	
11	Italy	Required for government contracts
12	Japan	Required for government contracts
13	Malaysia	Required for government contracts
14	Mexico	
15	Norway	
16	Peru	
17	Poland	
18	Singapore	
19	South Korea	Required for government contracts
20	Spain	
21	Switzerland	
22	Taiwan	
23	The Netherlands	
24	United Kingdom	
25	United States	

To be blunt, any company or government agency in the world that does not use function point metrics does not have accurate benchmark data on either quality or productivity. The software industry has had poor quality for over 50 years and a key reason for this problem is that the software industry has not measured quality well enough make effective improvements. Cost per defect and lines of code both distort reality and conceal progress. They are harmful rather than helpful in improving either quality or productivity. Lines of code reverses true economic productivity and makes assembly language seem more productive than Objective C. Cost per defect reverses true quality economics and makes buggy software look cheaper than high quality software. These distortions of economic reality have slowed software progress for over 50 years.

The U.S. industries that tend to use function point metrics and therefore understand software economics fairly well include automotive manufacturing, banks, commercial software, insurance, telecommunications, and some

public utilities. For example Bank of Montreal was one of the world's first users of function points after IBM placed the metric in the public domain; Ford has used function point metrics for fuel injection and navigation packages; Motorola has used function points for smart phone applications; AT&T has used function points for switching software; IBM has used function points for both commercial software and also operating systems. The U.S. industries that do not use function points widely and hence have no accurate data on either software quality or productivity include the Department of Defense, most state governments, the U.S. Federal government, and most universities (which should understand software economics but don't seem to.)

Although the Department of Defense was proactive in endorsing the Software Engineering Institute (SEI) capability maturity model integrated (CMMI), it lags the civilian sector in software metrics and measurements. For that matter the SEI itself has not yet supported function point metrics nor pointed out to clients that both lines of code and cost per defect distort reality and reverse the true economic value of high quality and high-level programming languages. It is interesting that the author had a contract from the U.S. Air Force to examine the benefits of ascending to the higher CMMI levels because the SEI itself had no quantitative data available. In fact the findings from this study are shown later in this report in Table 12. Although the Department of Defense itself lags in function point use some of the military services have used function points for important projects. For example the U.S. Navy has used function points for shipboard gun controls and cruise missile navigation. If a company or government agency wants to get serious in improving quality then the best and only effective metrics for achieving this are the combination of defect potentials in function points and defect removal efficiency (DRE).

Defect removal efficiency (DRE) is calculated by keeping accurate counts of all defects found during development. After release all customer-reported bugs are included in the total. After 90 days of customer usage DRE is calculated. If developers found 900 bugs and customer reported 50 bugs in the first three months then DRE is 95%. Obviously bug reports don't stop cold after 90 days, but the fixed 90-day interval provides an excellent basis for statistical quality reports. The overall range in defect potentials runs from about 2.00 per function point to more than 7.00 per function point. Factors that influence defect potentials include team skills, development methodologies, CMMI levels, programming languages, and defect prevention techniques such as joint application design (JAD) and quality function deployment (QFD). Some methodologies such as team software process (TSP) are "quality strong" and have low defect potentials.) Agile is average for defect potentials. Waterfall is worse than average for defect potentials. Table 4 shows the U.S. ranges for defect potentials circa 2016:

Table 4: U.S Average Ranges of Defect Potentials Circa 2016
(Defects per IFPUG 4.3 function point)

Defect Origins	Best	Average	Worst
Requirements	0.34	0.70	1.35
Architecture	0.04	0.10	0.20
Design	0.63	0.95	1.58
Code	0.44	1.15	2.63
Security flaws	0.18	0.25	0.40
Documents	0.20	0.45	0.54
Bad fixes	0.39	0.65	1.26
TOTAL	2.21	4.25	7.95

NOTE: the author's Software Risk Master (SRM) estimating tool predicts defect potentials as a standard output for every project estimated. Defect potentials obviously vary by size, with small projects typically having low defect potentials. Defect potentials rise faster than size increases, with large systems above 10,000 function points having alarmingly high defect potentials.

Table 5 shows U.S. ranges in defect potentials from small projects of 1 function point up to massive systems of 100,000 function points:

Table 5: Software Defect Potentials per Function Point by Size
(Defects per IFPUG 4.3 function point)

Function Points	Best	Average	Worst
1	0.60	1.50	2.55
10	1.25	2.50	4.25
100	1.75	3.25	6.13
1000	2.14	4.75	8.55
10000	3.38	6.50	12.03
100000	4.13	8.25	14.19
Average	2.21	4.25	7.95

As can be seen defect potentials go up rapidly with application size. This is one of the key reasons why large systems fail so often and also run late and over budget. Table 6 shows the overall U.S. ranges in defect removal efficiency (DRE) by applications size from a size of 1 function point up to 100,000 function points. As can be seen DRE goes down as size goes up:

Table 6: U.S. Software Average DRE Ranges by Application Size

Function Points	Best	Average	Worst
1	99.90%	97.00%	94.00%
10	99.00%	96.50%	92.50%
100	98.50%	95.00%	90.00%
1000	96.50%	94.50%	87.00%
10000	94.00%	89.50%	83.50%
100000	91.00%	86.00%	78.00%
Average	95.80%	92.20%	86.20%

Table 7 is a somewhat complicated table that combines the results of tables 5 and 6; i.e. both defect potentials and defect removal efficiency (DRE) ranges are now shown together on the same table. Note that as size increases defect potentials also increase, but defect removal efficiency (DRE) comes down:

Table 7: Software Defect Potentials and DRE Ranges by Size

Function Points		Best	Average	Worst
1	Defect Potential	0.60	1.50	2.55
	DRE	99.90%	97.00%	94.00%
	Delivered defects	0.00	0.05	0.15
10	Defect Potential	1.25	2.50	4.25

	DRE	99.00%	96.00%	92.50%
	Delivered defects	0.01	0.10	0.32
100	Defect Potential	1.75	3.50	6.13
	DRE	98.50%	95.00%	90.00%
	Delivered defects	0.03	0.18	0.61
1000	Defect Potential	2.14	4.75	8.55
	DRE	96.50%	94.50%	87.00%
	Delivered defects	0.07	0.26	1.11
10000	Defect Potential	3.38	6.50	12.03
	DRE	94.00%	89.50%	83.50%
	Delivered defects	0.20	0.68	1.98
100000	Defect Potential	4.13	8.25	14.19
	DRE	91.00%	86.00%	78.00%
	Delivered defects	0.37	1.16	3.12

Best-case results are usually found for software controlling medical devices or complex physical equipment such as aircraft navigation packages, weapons systems, operating systems, or telecommunication switching systems. These applications are usually large and range from about 1000 to over 100,000 function points in size. Large complex applications require very high DRE levels in order for the physical equipment to operate safely. They normally use pre-test inspections and static analysis and usually at least 10 test stages. **Average-case** results are usually found among banks, insurance companies, manufacturing, and commercial software. These applications are also on the large size and range from 1000 to more than 10,000 function points. Here too high levels of DRE are important since these applications contain and deal with confidential data. These applications normally use pre-test static analysis and at least 8 test stages.

Worst-case results tend to show up in litigation for cancelled projects or for lawsuits for poor quality. State, municipal, and civilian Federal government software projects, and especially large systems such as a taxation, child support, and motor vehicles are often in the worst-case class. It is an interesting point that every lawsuit where the author has worked as an expert witness has been for large systems > 10,000 function points in size. These applications seldom use either pre-test inspections or static analysis and sometimes use only 6 test stages. While function point metrics are the best choice for normalization, it is also important to know the actual numbers of defects that are likely to be present when software applications are delivered to customers. Table 8 shows data from table 7 only expanded to show total numbers of delivered defects:

Table 8: U.S. Average Delivered Defects by Application Size

Function Points	Best	Average	Worst
1	0	0	1
10	0	1	3
100	3	18	61
1000	75	261	1,112
10000	2,028	6,825	19,841
100000	3,713	11,550	31,218
Average	970	3,109	8,706

Here too it is painfully obvious that defect volumes go up with application size. However table 8 shows all severity levels of delivered defects. Only about 1% of delivered defects will be in the high-severity class of 1 and only about 14% in severity class 2. Severity class 3 usually has about 55% while severity 4 has about 30%. Defect potentials have also varied by decade. Table 9 shows approximate values starting in 1960 and ending with projected values for 2019. The reason for the gradual improvement in defect potentials include the advent of newer programming languages, the average increase in organizations with higher CMMI levels, a gradual decrease in application size, and a gradual increase on reusable materials from older applications.

Table 9: Defect Potentials by Decade

	Best	Average	Worst
1960-1969	2.85	5.50	10.29
1970-1979	2.72	5.25	9.82
1980-1989	2.59	5.00	9.35
1990-1999	2.46	4.75	8.88
2000-2009	2.33	4.50	8.42
2010-2019	2.20	4.25	7.95

These severity levels are normally assigned by software quality assurance personnel. Because companies fix high severity bugs faster than low severity bugs, clients often report bugs as being severity 2 that are really only severity 3 or severity 4. While the IBM average for severity 2 bugs was about 14%, clients tend to exaggerate and rank over 50% of bug reports as severity 2!

This classification of defect severity levels was developed by IBM circa 1960: It has been used for over 50 years by thousands of companies for hundreds of thousands of software applications.

Table 10: IBM Defect Severity Scale (1960 – 2016)

Severity 1	Software does not work at all
Severity 2	Major features disabled and inoperative
Severity 3	Minor bug that does not prevent normal use
Severity 4	Cosmetic errors that do not affect operation
Invalid	Defects not correctly reported; i.e. hardware problems reported as software
Duplicate	Multiple reports of the same bug
Abeyant	Unique defects found by only 1 client that cannot be duplicated

It is obvious that valid high-severity defects of severities 1 and 2 are the most troublesome for software projects. Defect removal efficiency (DRE) is a powerful and useful metric. Every important project should measure DRE and every important project should top 99% in DRE, but few do. As defined by IBM circa 1973 DRE is measured by keeping track of all bugs found internally during development, and comparing these to customer-reported bugs during the first 90 days of usage. If internal bugs found during development total 95 and customers report 5 bugs in the first three months of use then DRE is 95%.

Another important quality topic is that of “error-prone modules” (EPM) also discovered by IBM circa 1970. IBM did a frequency analysis of defect distributions and was surprised to find that bugs are not randomly distributed, but clump in a small number of modules. For example in the IBM IMS data base application there were 425

modules. About 300 of these were zero-defect modules with no customer-reported bugs. About 57% of all customer reported bugs were noted in only 31 modules out of 425. These tended to be high in cyclomatic complexity, and also had failed to use pre-test inspections. Table 11 shows approximate results for EPM in software by application size:

Table 11: Distribution of "Error Prone Modules" (EPM) in Software

Function Points	Best	Average	Worst
1	0	0	0
10	0	0	0
100	0	0	0
1000	0	2	4
10000	0	18	49
100000	0	20	120
Average	0	7	29

EPM were discovered by IBM but unequal distribution of bugs was also noted by many other companies whose defect tracking tools can highlight bug reports by modules. For example EPM were confirmed by AT&T, ITT, Motorola, Boeing, Raytheon, and other technology companies with detailed defect tracking systems. EPM tend to resist testing, but are fairly easy to find using pre-test static analysis, pre-test inspections, or both. EPM are treatable, avoidable conditions and should not be allowed to occur in modern software circa 2016. The presence of EPM is a sign of inadequate defect quality measurements and inadequate pre-test defect removal activities.

The author had a contract from the U.S. Air Force to examine the value of ascending to the higher levels of the capability maturity model integrated (CMMI). Table 12 shows the approximate quality results for all five levels of the CMMI:

Table 12: Software Quality and the SEI Capability Maturity Model Integrated (CMMI) for 2,500 function points

CMMI Level	Defect Potential per Function Point	Defect Removal Efficiency	Delivered Defects per Function Point	Delivered Defects
SEI CMMI 1	4.50	87.00%	0.585	1,463
SEI CMMI 2	3.85	90.00%	0.385	963
SEI CMMI 3	3.00	96.00%	0.120	300
SEI CMMI 4	2.50	97.50%	0.063	156
SEI CMMI 5	2.25	99.00%	0.023	56

Table 12 was based on study by the author commissioned by the U.S. Air Force. Usage of the CMMI is essentially limited to military and defense software. Few civilian companies use the CMMI and the author has met several CIO's from large companies and state governments that have never even heard of SEI or the CMMI. Software defect potentials and DRE also vary by industry. Table 13 shows a sample of 15 industries with higher than average quality levels out of a total of 75 industries where the author has data:

Table 13: Software Quality Results by Industry

		Defect Potentials per Function Point 2016	Defect Removal Efficiency 2016	Delivered Defects per Function Pt 2016
Industry				
Best Quality				
1	Manufacturing - medical devices	4.60	99.50%	0.02
2	Manufacturing - aircraft	4.70	99.00%	0.05
3	Government - military	4.70	99.00%	0.05
4	Smartphone/tablet applications	3.30	98.50%	0.05
5	Government - intelligence	4.90	98.50%	0.07
6	Software (commercial)	3.50	97.50%	0.09
7	Telecommunications operations	4.35	97.50%	0.11
8	Manufacturing - defense	4.65	97.50%	0.12
9	Manufacturing - telecommunications	4.80	97.50%	0.12
10	Process control and embedded	4.90	97.50%	0.12
11	Manufacturing - pharmaceuticals	4.55	97.00%	0.14
12	Professional support - medicine	4.80	97.00%	0.14
13	Transportation - airlines	5.87	97.50%	0.15
14	Manufacturing - electronics	4.90	97.00%	0.15
15	Banks - commercial	4.15	96.25%	0.16

There are also significant differences by country. Table 14 shows a sample of 15 countries with better than average quality out of a total of 70 countries where the author has data:

Table 14: Samples of Software Quality by Country

		Defect Potential per FP 2016	Defect Removal Efficiency (DRE) 2016	Delivered Defects per Function Pt 2016
Countries				
Best Quality				
1	Japan	4.25	96.00%	0.17
2	India	4.90	95.50%	0.22
3	Finland	4.40	94.50%	0.24
4	Switzerland	4.40	94.50%	0.24
5	Denmark	4.25	94.00%	0.26
6	Israel	5.00	94.80%	0.26
7	Sweden	4.45	94.00%	0.27
8	Netherlands	4.40	93.50%	0.29
9	Hong Kong	4.45	93.50%	0.29
10	Brazil	4.50	93.00%	0.32
11	Singapore	4.80	93.40%	0.32
12	United Kingdom	4.55	93.00%	0.32
13	Malaysia	4.60	93.00%	0.32
14	Norway	4.65	93.00%	0.33
15	Taiwan	4.90	93.30%	0.33

Countries such as Japan and India tend to be more effective in pre-test defect removal operations and to use more certified test personnel than those lower down the table. Although not shown in table 14 the U.S. ranks as country #19 out of the 70 countries from which the author has data. Table 15 shows quality comparison of 15 software development methodologies (this table is cut down from a larger table of 80 methodologies that will be published in the author's next book.)

Table15: Comparisons of 15 Software Methodologies

Methodologies		Defect Potential per FP 2016	Defect Removal Efficiency 2016	Delivered Defects per FP 2016
Best Quality				
1	Reuse-oriented (85% reusable materials)	1.30	99.50%	0.007
2	Pattern-based development	1.80	99.50%	0.009
3	Animated, 3D, full color design development	1.98	99.20%	0.016
4	Team software process (TSP) + PSP	2.35	98.50%	0.035
5	Container development (65% reuse)	2.90	98.50%	0.044
6	Microservice development	2.50	98.00%	0.050
7	Model-driven development	2.60	98.00%	0.052
8	Microsoft SharePoint development	2.70	97.00%	0.081
9	Mashup development	2.20	96.00%	0.088
10	Product Line engineering	2.50	96.00%	0.100
11	DevOps development	3.00	94.00%	0.180
12	Pair programming development	3.10	94.00%	0.186
13	Agile + scrum	3.20	92.50%	0.240
14	Open-source development	3.35	92.00%	0.268
15	Waterfall development	4.60	87.00%	0.598

Table 16 shows the details of how defect removal efficiency (DRE) operates. Table 16 must of course use fixed values but there are ranges for every row and column for both pre-test and test methods.

There are also variations in the numbers of pre-test removal and test stages used. Table 16 illustrates the maximum number observed.

The data in table 16 is originally derived from IBM's software quality data collection which is more complete than most companies. Other companies have been studied as well. Note that requirements defects are among the most difficult to remove since they are resistant to testing.

To consistently top 99% in DRE the minimum set of methods needed include most of the following:

Pre-Test Removal

1. Formal Inspections (requirements, design, code, etc.)
2. Code Static analysis
3. Automated Requirements modeling
4. Automated correctness proofs

Test Removal

1. Unit test (manual/automated)
2. Function test
3. Regression test
4. Integration test

5. Performance test
6. Usability test
7. Security test
8. System test
9. Field or acceptance test

In other words a series of about 13 kinds of defect removal activities are generally needed to top 99% in DRE consistently. Testing by itself without inspections or static analysis usually is below 90% in DRE.

Of course some critical applications such as medical devices and weapons systems use many more kinds of testing. As many as 18 kinds of testing have been observed by the author. This paper uses 12 kinds of testing since these are fairly common on large systems > 10,000 function points in size which is where quality is a critical factor.

Note that DRE includes bugs that originate in architecture, requirements, design, code, documents, and "bad fixes" or new bugs in bug repairs themselves. All bug origins should be included since requirements and design bugs often outnumber code bugs.

Note that the defect potential for next table 16 is somewhat lower than the 4.25 value shown in tables 1, 2, and 3. This is because those tables includes all programming languages and some have higher defect potentials than Java, which is used for table 16.

Code defect potentials vary by language with low-level languages such as assembly and C having a higher defect potential than high-level languages such as Java, Objective C, C#, Ruby, Python, etc.

Table16: Software Quality and Defect Removal Efficiency (DRE)

Note 1: The table represents high quality defect removal operations.

Application size in function points		1,000				
Application language		Java				
Source lines per FP		53.33				
Source lines of code		53,330				
Pre-Test Defect Removal Methods	Architect. Defects per Function Point	Require. Defects per Function Point	Design Defects per Function Point	Code Defects per Function Point	Document Defects per Function Point	TOTALS
Defect Potentials per Function Point	0.25	1.00	1.15	1.30	0.45	4.15
Defect potentials	250	1,000	1,150	1,300	450	4,150
1 Requirement inspection	5.00%	87.00%	10.00%	5.00%	8.50%	26.52%
Defects discovered	13	870	115	65	38	1,101
Bad-fix injection	0	26	3	2	1	33
Defects remaining	237	104	1,032	1,233	411	3,016
2 Architecture inspection	85.00%	10.00%	10.00%	2.50%	12.00%	13.10%
Defects discovered	202	10	103	31	49	395
Bad-fix injection	6	0	3	1	1	12
Defects remaining	30	93	925	1,201	360	2,609
3 Design inspection	10.00%	14.00%	87.00%	7.00%	16.00%	36.90%
Defects discovered	3	13	805	84	58	963

	Bad-fix injection	0	0	24	3	2	48
	Defects remaining	26	80	96	1,115	301	1,618
4	Code inspection	12.50%	15.00%	20.00%	85.00%	10.00%	62.56%
	Defects discovered	3	12	19	947	30	1,012
	Bad-fix injection	0	0	1	28	1	30
	Defects remaining	23	67	76	139	270	575
5	Code Static Analysis	2.00%	2.00%	7.00%	55.00%	3.00%	15.92%
	Defects discovered	0	1	5	76	8	92
	Bad-fix injection	0	0	0	2	0	3
	Defects remaining	23	66	71	60	261	481
6	IV & V	10.00%	12.00%	23.00%	7.00%	18.00%	16.16%
	Defects discovered	2	8	16	4	47	78
	Bad-fix injection	0	0	0	0	1	2
	Defects remaining	20	58	54	56	213	401
7	SQA review	10.00%	17.00%	17.00%	12.00%	12.50%	30.06%
	Defects discovered	2	10	9	7	27	54
	Bad-fix injection	0	0	0	0	1	3
	Defects remaining	18	48	45	49	185	344
	Pre-test defects removed	232	952	1,105	1,251	265	3,805
	Pre-test efficiency %	92.73%	95.23%	96.12%	96.24%	58.79%	91.69%
	Test Defect Removal Stages						
		Architect.	Require.	Design	Code	Document	Total
1	Unit testing (Manual)	2.50%	4.00%	7.00%	35.00%	10.00%	11.97%
	Defects discovered	0	2	3	17	19	41
	Bad-fix injection	0	0	0	1	1	1
	Defects remaining	18	46	41	31	166	301
2	Function testing	7.50%	5.00%	22.00%	37.50%	10.00%	13.63%
	Defects discovered	1	2	9	12	17	41
	Bad-fix injection	0	0	0	0	0	1
	Defects remaining	16	43	32	19	149	259
3	Regression testing	2.00%	2.00%	5.00%	33.00%	7.50%	7.84%
	Defects discovered	0	1	2	6	11	20
	Bad-fix injection	0	0	0	0	0	1
	Defects remaining	16	43	30	13	138	238
4	Integration testing	6.00%	20.00%	22.00%	33.00%	15.00%	17.21%
	Defects discovered	1	9	7	4	21	41
	Bad-fix injection	0	0	0	0	1	1
	Defects remaining	15	34	23	8	116	196
5	Performance testing	14.00%	2.00%	20.00%	18.00%	2.50%	6.07%
	Defects discovered	2	1	5	2	3	12
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	13	33	19	7	113	184

6	Security testing	12.00%	15.00%	23.00%	8.00%	2.50%	7.71%
	Defects discovered	2	5	4	1	3	14
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	11	28	14	6	110	169
7	Usability testing	12.00%	17.00%	15.00%	5.00%	48.00%	36.42%
	Defects discovered	1	5	2	0	53	62
	Bad-fix injection	0	0	0	0	2	2
	Defects remaining	10	23	12	6	56	106
8	System testing	16.00%	12.00%	18.00%	12.00%	34.00%	24.81%
	Defects discovered	2	3	2	1	19	26
	Bad-fix injection	0	0	0	0	1	1
	Defects remaining	8	20	10	5	36	79
9	Cloud testing	10.00%	5.00%	13.00%	10.00%	20.00%	13.84%
	Defects discovered	1	1	1	1	7	11
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	7	19	8	5	29	69
10	Independent testing	12.00%	10.00%	11.00%	10.00%	23.00%	15.81%
	Defects discovered	1	2	1	0	7	11
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	6	17	8	4	22	57
11	Field (Beta) testing	14.00%	12.00%	14.00%	12.00%	34.00%	20.92%
	Defects discovered	1	2	1	1	7	12
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	6	15	6	4	14	45
12	Acceptance testing	13.00%	14.00%	15.00%	12.00%	24.00%	20.16%
	Defects discovered	1	2	1	0	6	10
	Bad-fix injection	0	0	0	0	0	0
	Defects remaining	5	13	6	3	8	35
	Test Defects Removed	13	35	39	46	177	309
	Testing Efficiency %	73.96%	72.26%	87.63%	93.44%	95.45%	89.78%
	Total Defects Removed	245	987	1,144	1,297	442	4,114
	Total Bad-fix injection	7	30	34	39	13	123
	Cumulative Removal %	98.11%	98.68%	99.52%	99.75%	98.13%	99.13%
	Remaining Defects	5	13	6	3	8	36
	High-severity Defects	1	2	1	1	1	5
	Security Defects	0	0	0	0	0	1
	Remaining Defects per Function Point	0.0036	0.0102	0.0042	0.0025	0.0065	0.0278
	Remaining Defects per K Function Points	3.63	10.17	4.23	2.46	6.48	27.81
	Remaining Defects per KLOC	0.09	0.25	0.10	0.06	0.16	0.68

Note: The letters “IV&V” in table 16 stand for “independent verification and validation.” This is a method used by defense software projects but it seldom occurs in the civilian sector. The efficiency of IV&V is fairly low and the costs are fairly high. DRE measures can be applied to any combination of pre-test and testing stages. Table 16 shows seven pre-test DRE activities and 12 kinds of testing: 19 forms of defect removal in total. This combination would only be used on large defense systems and also on critical medical devices. It might also be used on aircraft navigation and avionics packages. In other words software that might cause injury or death to humans if quality lags are the most likely to use both DRE measures and sophisticated combinations of pre-test and test removal methods.

As of 2016 the U.S. average for DRE is only about 92.50%. This is close to the average for Agile projects. The U.S. norm is to use only static analysis before testing and six kinds of testing: unit test, function test, regression test, performance test, system test, and acceptance test. This combination usually results in about 92.50% DRE. If static analysis is omitted and only six test stages are used, DRE is normally below 85%. In this situation quality problems are numerous. Note that when a full suite of pre-test defect removal and test stages are used, the final number of defects released to customers often has more bugs originating in requirements and design than in code. Due to static analysis and formal testing by certified test personnel, DRE for code defects can top 99.75%. It is harder to top 99% for requirements and design bugs since both resist testing and can only be found via inspections, or by text static analysis.

Software Quality and Software Security

Software quality and software security have a tight relationship. Security flaws are just another kind of defect potential. As defect potentials go up so do security flaws, as DRE declines more and more security flaws will be released.

Of course security has some special methods that are not part of traditional quality assurance. One of these is the use of ethical hackers and another is the use of penetration teams that deliberately try to penetrate the security defenses of critical software applications.

Security also includes social and physical topics that are not part of ordinary software operations. For example security requires careful vetting of personnel. Security for really critical applications may also require Faraday cages around computers to ensure that remote sensors are blocked and can’t steal information from a distance or through building walls.

To provide an approximate set of values for high-severity defects and security flaws table 16 shows what happens when defect potentials increase and DRE declines. To add realism to this example table 17 uses a fixed size of 1000 function points. Delivered defects, high-severity defects, and security flaws are shown in whole numbers rather than defects per function point:

Table 17: Quality and Security Flaws for 1000 Function Points

Defect Potentials per FP	DRE	Delivered Defects per FP	Delivered Defects	High Severity Defects	Security Flaw Defects
2.50	99.50%	0.01	13	1	0
3.00	99.00%	0.03	30	3	0
3.50	97.00%	0.11	105	10	1
4.00	95.00%	0.20	200	21	3
4.25	92.50%	0.32	319	35	4
4.50	92.00%	0.36	360	42	6
5.00	87.00%	0.65	650	84	12
5.50	83.00%	0.94	935	133	20
6.00	78.00%	1.32	1,320	206	34

The central row in the middle of this table highlighted in blue show approximate 2016 U.S. averages in terms of delivered defects, high-severity defects, and latent security flaws for 1000 function points. The odds of a successful cyber-attack would probably be around 15%. At the safe end of the spectrum where defect potentials are low and DRE tops 99% the number of latent security flaws is 0. The odds of a successful cyber-attack are very low at the safe end of the spectrum: probably below 1%. At the dangerous end of the spectrum with high defect potentials and low DRE, latent security flaws top 20 for 1000 function points. This raises the odds of a successful cyber-attack to over 50%.

Software Quality and Technical Debt

Ward Cunningham introduced an interesting metaphor called “technical debt” which concerns latent defects present in software applications after deployment. The idea of technical debt is appealing but unfortunately technical debt is somewhat ambiguous and every company tends to accumulate data using different methods so it is hard to get accurate benchmarks. In general technical debt deals with the direct costs of fixing latent defects as they are reported by users or uncovered by maintenance personnel. However there are other and larger costs associated with legacy software and also new software that are not included in technical debt:

1. Litigation against software outsource contractors or commercial software vendors by disgruntled users who sue for excessive defects.
2. Consequential damages or financial harm to users of defective software. For example if the computerized brake system of an automobile fails and causes a serious accident, neither the cost of repairing the auto nor any medical bills for injured passengers are included in technical debt.
3. Latent security flaws that are detected by unscrupulous organizations and lead to data theft, denial of service, or other forms of cyber-attack are not included in technical debt either.

Technical debt is an appealing metaphor but until consistent counting rules become available it is not a satisfactory quality metric. The author suggests that the really high cost topics of consequential damages, cyber-attacks, and litigation for poor quality should be included in technical debt or at least not ignored as they are in 2016.

Assume a software outsource vendor builds a 10,000 function point application for a client for a cost of \$30,000,000 and it has enough bugs to make the client unhappy. True technical debt or the costs of repairing latent defects found and reported by clients over several years after deployment might cost about \$5,000,000.

However depending upon what the application does, consequential damages to the client could top \$25,000,000; litigation by the unhappy client might cost \$5,000,000; severe cyber-attacks and data theft might cost \$30,000,000: a total cost of \$60,000,000 over and above the nominal amount for technical debt.

Of these problems cyber-attacks are the most obvious candidates to be added to technical debt because they are the direct result of latent security flaws present in the software when it was deployed. The main difference between normal bugs and security flaws is that cyber criminals can exploit security flaws to do very expensive damages to software (and even hardware) or to steal valuable and sometimes classified information.

In other words possible post-release costs due to poor quality control might approach or exceed twice the initial costs of development; and 12 times the costs of “technical debt” as it is normally calculated.

SNAP Metrics for Non-Functional Size

In 2011 the IFPUG organization developed a new metric for non-functional requirements. This metric is called “SNAP” which is sort of an acronym for “software non-functional assessment process.” (No doubt future sociologists will puzzle over software naming conventions.) Unfortunately the SNAP metric was not created to be equivalent to standard IFPUG function points. That means if you have 100 function points and 15 SNAP points you cannot add them together to create 115 total “points.” This makes both productivity and quality studies more difficult because function point and SNAP work needs to be calculated separately. Since one of the

most useful purposes for function point metrics has been for predicting and measuring quality, the addition of SNAP metrics to the mix has raised the complexity of quality calculations. Pasted below are the results of an experimental quality calculation tool developed by the author that can combine defect potentials and defect removal efficiency (DRE) for both function point metrics and the newer SNAP metrics.

Table 18: SNAP Software Defect Calculator

6/9/2016

I

Size in Function Points **1,000**
Size in SNAP Points **152**

Defect Origins	Defects Per FP	Defects per SNAP	SNAP Percent
Requirements	0.70	0.14	19.50%
Architecture	0.10	0.02	15.50%
Design	0.95	0.18	18.50%
Source code	1.15	0.13	11.50%
Security flaws	0.25	0.05	20.50%
Documents	0.45	0.02	3.50%
Bad Fixes	0.65	0.12	18.50%
TOTALS	4.25	0.65	15.23%

Defect Origins	Defect Potential	Defects Potential
Requirements	700	21
Architecture	100	2
Design	950	27
Source code	1,150	20
Security flaws	250	8
Documents	450	2
Bad Fixes	650	18
TOTALS	4,250	99

Defect Origins	Removal Percent	Removal Percent
Requirements	75.00%	75.00%
Architecture	70.00%	70.00%
Design	96.00%	96.00%

Source code	98.00%	98.00%
Security flaws	87.00%	87.00%
Documents	95.00%	95.00%
Bad Fixes	78.00%	78.00%
Average	85.57%	85.57%

Defect Origins	Delivered Defects	Delivered Defects
Requirements	175	5
Architecture	30	1
Design	38	1
Source code	23	0
Security flaws	33	1
Documents	23	0
Bad Fixes	143	4
Total	464	13

Defect Origins	Delivered Per FP	Delivered per SNAP	SNAP Percent
Requirements	0.175	0.034	19.50%
Architecture	0.030	0.005	15.50%
Design	0.038	0.007	18.50%
Source code	0.023	0.003	11.50%
Security flaws	0.023	0.007	29.61%
Documents	0.143	0.026	18.50%
Bad Fixes	0.464	0.082	17.75%
Total	0.896	0.164	18.31%

In real life defect potentials go up with application size and defect removal efficiency (DRE) comes down with application size. This experimental tool holds defect potentials and DRE as constant values. The purpose is primarily to experiment with the ratios of SNAP defects and with DRE against SNAP bugs.

A great deal more study and more empirical data is needed before SNAP can actually become useful for software quality analysis. Right now there is hardly any empirical data available on SNAP and software quality.

Economic Value of High Software Quality

One of the major economic weaknesses of the software industry due to bad metrics and poor measurements is a total lack understanding of the economic value of high software quality. If achieving high quality levels added substantially to development schedules and development costs it might not be worthwhile to achieve it. But the good news is that high software quality levels comes with shorter schedules and lower costs than average or poor quality! These reductions in schedules and costs, or course, are due to the fact that finding and fixing bugs has been the #1 software cost driver for over 50 years. When defect potentials are reduced and DRE is increased due to pre-test defect removal such as static analysis, then testing time and testing costs shrink dramatically.

Table 19 shows the approximate schedules in calendar months, the approximate effort in work hours per function point, and the approximate \$ cost per function point that results from various combinations of software defect potentials and defect removal efficiency.

The good news for the software industry is that low defect potentials and high DRE levels are the fastest and cheapest way to build software applications!

Table 19: Schedules, Effort, Costs for 1000 Function Points
(Monthly costs = \$10,000)

Defect Potentials per FP	DRE	Delivered Defects per FP	Delivered Defects	Schedule Months	Work Hours per Function Point	Development Cost per Function Point	\$ per Defect (Caution!)
2.50	99.50%	0.01	13	13.34	12.00	\$909.09	\$4,550.00
3.00	99.00%	0.03	30	13.80	12.50	\$946.97	\$3,913.00
3.50	97.00%	0.11	105	14.79	13.30	\$1,007.58	\$3,365.18
4.00	95.00%	0.20	200	15.85	13.65	\$1,034.09	\$2,894.05
4.25	92.50%	0.32	319	16.00	13.85	\$1,050.00	\$2,488.89
4.50	92.00%	0.36	360	16.98	14.00	\$1,060.61	\$2,140.44
5.00	87.00%	0.65	650	18.20	15.00	\$1,136.36	\$1,840.78
5.50	83.00%	0.94	935	19.50	16.50	\$1,250.00	\$1,583.07
6.00	78.00%	1.32	1,320	20.89	17.00	\$1,287.88	\$1,361.44

The central row highlighted in blue shows approximate U.S. average values for 2016. This table also shows the “cost per defect” metric primarily to caution readers that this metric is inaccurate and distorts reality since it make buggy applications look cheaper than high-quality applications.

A Primer on Manufacturing Economics and the Impact of Fixed Costs

The reason for the distortion of the cost per defect metric is because cost per defect ignores the fixed costs of writing test cases, running test cases, and for maintenance the fact that the change team must be ready whether or not bugs are reported.

To illustrate the problems with the cost per defect metric, assume you have data on four identical applications of 1000 function points in size. Assume for all four that writing test cases costs \$10,000 and running test cases costs \$10,000 so fixed costs are \$20,000 for all four cases.

Now assume that fixing bugs costs exactly \$500 each for all four cases. Assume Case 1 found 100 bugs, Case 2 found 10 bugs, Case 3 found 1 bug, and Case 4 had zero defects with no bugs found by testing. Table 20 illustrates both cost per defect and cost per function point for these four cases:

Table 20: Comparison of \$ per defect and \$ per function point

	Case 1	Case 2	Case 3	Case 4
Fixed costs	\$20,000	\$20,000	\$20,000	\$20,000
Bug repairs	\$50,000	\$5,000	\$500	\$0
Total costs	\$70,000	\$25,000	\$20,500	\$20,000

Bugs found	100	10	1	0
\$ per defect	\$700	\$2,500	\$20,500	Infinite
\$ per FP	\$70.00	\$25.00	\$20.50	\$20.00

As can be seen the “cost per defect” metric penalizes quality and gets more expensive as defect volumes decline. This is why hundreds of refereed papers all claim that cost per defect goes up later in development. The real reason that cost per defect goes up is not that the actual cost of defect repairs goes up, but rather fixed costs make it look that way. Cost per function point shows the true economic value of high quality and this goes down as defects decline.

Recall a basic law of manufacturing economics that “If a manufacturing process has a high percentage of fixed costs and there is a decline in the number of units produced, the cost per unit will go up.” For over 50 years the cost per defect metric has distorted reality and concealed the true economic value of high quality software. Some researchers have suggested leaving out the fixed costs of writing and running test cases and only considering the variable costs of actual defect repairs. This violates both economic measurement principles and also and good sense. Would you want a contractor to give you an estimate for building a house that only showed foundation and framing costs but not the more variable costs of plumbing, electrical wiring, and internal finishing? Software Cost of Quality (COQ) needs to include ALL of the cost elements of finding and fixing bugs and not just a small subset of those costs.

The author has read over 100 refereed software articles in major journals such as IEEE Transactions, IBM Systems Journal, Cutter Journal, and others that parroted the stock phrase *“It costs 100 times more to fix a bug after release than it does early in development.”* Not even one of these 100 articles identified the specific activities that were included in the cost per defect data. Did the authors include test case design, test case development, test execution, defect logging, defect analysis, inspections, desk checking, correctness proofs, static analysis, all forms of testing, post-release defects, abeyant defects, invalid defects, duplicate defects, bad fix injections, error-prone modules or any of the other topics that actually have a quantified impact on defect repairs?

Not even one of the 100 journal articles included such basic information on the work elements that comprised the “cost per defect” claims by the authors. In medical journals this kind of parroting of a stock phrase without defining any of its elements would be viewed as professional malpractice. But the software literature is so lax and so used to bad data, bad metrics, and bad measures that none of the referees probably even noticed that the cost per defect claims were unsupported by any facts at all. The omission of fixed costs also explains why “lines of code” metrics are invalid and penalize high-level languages. In the case of LOC metrics requirements, design, architecture, and other kinds of non-code work are fixed costs, so when there is a switch from a low-level language such as assembly to a higher level language such as Objective C the “cost per line of code” goes up.

Table 21 shows 15 programming languages with cost per function point and cost per line of code in side by side columns, to illustrate that LOC penalizes high-level programming languages, distorts reality, and reverses the true economic value of high-level programming languages:

Table 21: Productivity Expressed Using both LOC and Function Points

	Languages	Size in LOC	Coding Work hrs	Total Work hrs	Total Costs	\$ per FP	\$ per LOC
1	Application Generators	7,111	1,293	4,293	\$325,222	\$325.22	\$45.73
2	Mathematica10	9,143	1,662	4,662	\$353,207	\$353.21	\$38.63

3	Smalltalk	21,333	3,879	6,879	\$521,120	\$521.12	\$24.43
4	Objective C	26,667	4,848	7,848	\$594,582	\$594.58	\$22.30
5	Visual Basic	26,667	4,848	7,848	\$594,582	\$594.58	\$22.30
6	APL	32,000	5,818	8,818	\$668,044	\$668.04	\$20.88
7	Oracle	40,000	7,273	10,273	\$778,237	\$778.24	\$19.46
8	Ruby	45,714	8,312	11,312	\$856,946	\$856.95	\$18.75
9	Simula	45,714	8,312	11,312	\$856,946	\$856.95	\$18.75
10	C#	51,200	9,309	12,309	\$932,507	\$932.51	\$18.21
11	ABAP	80,000	14,545	17,545	\$1,329,201	\$1,329.20	\$16.62
12	PL/I	80,000	14,545	17,545	\$1,329,201	\$1,329.20	\$16.62
13	COBOL	106,667	19,394	22,394	\$1,696,511	\$1,696.51	\$15.90
14	C	128,000	23,273	26,273	\$1,990,358	\$1,990.36	\$15.55
15	Macro Assembly	213,333	38,788	41,788	\$3,165,748	\$3,165.75	\$14.84

Recall that the standard economic definition for productivity for more than 200 years has been “Goods or services produced per unit of labor or expense.” If a line of code is selected as a unit of expense then moving to a high-level programming language will drive up the cost per LOC because of the fixed costs of non-code work.

Function point metrics, on the other hand, do not distort reality and are a good match to manufacturing economics and also to standard economics because they correctly show that the least expensive version has the highest economic productivity. LOC metrics make the most expensive version seem to have higher productivity than the cheapest, which of course violates standard economics. Also, software has a total of 126 occupation groups. The only occupation that can be measured at with “lines of code” is that of programming. Function point metrics, on the other hand, can measure the productivity of non-code occupations such as business analysts, architects, data base designers, technical writers, project management and everybody else. The author is often asked questions such as “*If cost per defect and lines of code are such bad metrics why do so many companies still use them?*” The questioners are assuming, falsely, that if large numbers of people do something it must be beneficial. There is no real correlation between usage and benefits. Usually it is only necessary to pose a few counter questions:

“If obesity is harmful why are so many people overweight?”

“If tobacco is harmful why do so many people smoke?”

As will be shown later in this report the number of users of the very harmful anti-pattern development methodology outnumber the users of the very beneficial pattern-based development methodology. There is very poor correlation between value and numbers of users. Many harmful things have thousands of users. The reason for continued usage of bad metrics is “cognitive dissonance” which is a psychological topic studied by Dr. Leon Festinger and first published in 1962. Today there is an extensive literature on cognitive dissonance. Dr. Festinger studied opinion formation and found that once an idea is accepted by the human mind, it is locked in place and won’t change until evidence against the idea is overwhelming. Then there will be an abrupt change to a new idea. Cognitive dissonance has been a key factor for resistance to many new innovations and new scientific theories including:

- Resistance to the theories of Copernicus and Galileo.
- Resistance to Lister’s and Semmelweis’s proposals for sterile surgical procedures.
- Resistance to Alfred Wegener’s theory of continental drift.
- Resistance to Charles Darwin’s theory of evolution.
- British naval resistance to self-leveling shipboard naval cannons.
- Union and Confederate Army resistance to replacing muskets with rifles.

- Naval resistance to John Ericsson's inventions of screw propellers and iron-clad ships.
- Army resistance to Christie's invention of military tank treads.
- Military and police resistance to Samuel Colt's revolvers (he went bankrupt.)
- Military resistance and the court martial of Gen. Billy Mitchell for endorsing air power.

Cognitive dissonance is a powerful force that has slowed down acceptance of many useful technologies. Table 22 illustrates the use of function points for 40 software development activities. It is obvious that serious software economic analysis needs to use activity-based costs and not just use single-point measures or phase-based measures neither of which can be validated.

Table 22: Function Points for Activity-Based Cost Analysis for 10,000 Function Points

		Work	Burdened		
		Hours per	Cost per	Project	% of
Development Activities	Funct. Pt.	Funct. Pt.	Cost	Cost	Total
1	Business analysis	0.01	\$0.42	\$4,200	0.02%
2	Risk analysis/sizing	0.00	\$0.14	\$1,400	0.01%
3	Risk solution planning	0.00	\$0.21	\$2,100	0.01%
4	Requirements	0.29	\$23.33	\$233,333	1.36%
5	Requirement. Inspection	0.24	\$19.09	\$190,909	1.11%
6	Prototyping	0.38	\$30.00	\$30,000	0.17%
7	Architecture	0.05	\$4.20	\$42,000	0.24%
8	Architecture. Inspection	0.04	\$3.00	\$30,000	0.17%
9	Project plans/estimates	0.04	\$3.00	\$30,000	0.17%
10	Initial Design	0.66	\$52.50	\$525,000	3.06%
11	Detail Design	0.88	\$70.00	\$700,000	4.08%
12	Design inspections	0.53	\$42.00	\$420,000	2.45%
13	Coding	6.60	\$525.00	\$5,250,000	30.58%
14	Code inspections	3.30	\$262.50	\$2,625,000	15.29%
15	Reuse acquisition	0.00	\$0.14	\$1,400	0.01%
16	Static analysis	0.01	\$0.70	\$7,000	0.04%
17	COTS Package purchase	0.01	\$0.42	\$4,200	0.02%
18	Open-source acquisition.	0.00	\$0.21	\$2,100	0.01%
19	Code security audit.	0.07	\$5.25	\$52,500	0.31%
20	Ind. Verif. & Valid. (IV&V)	0.01	\$1.05	\$10,500	0.06%
21	Configuration control.	0.03	\$2.10	\$21,000	0.12%
22	Integration	0.02	\$1.75	\$17,500	0.10%
23	User documentation	0.26	\$21.00	\$210,000	1.22%
24	Unit testing	1.06	\$84.00	\$840,000	4.89%
25	Function testing	0.94	\$75.00	\$750,000	4.37%
26	Regression testing	1.47	\$116.67	\$1,166,667	6.80%
27	Integration testing	1.06	\$84.00	\$840,000	4.89%
28	Performance testing	0.26	\$21.00	\$210,000	1.22%
29	Security testing	0.38	\$30.00	\$300,000	1.75%
30	Usability testing	0.22	\$17.50	\$175,000	1.02%
31	System testing	0.75	\$60.00	\$600,000	3.49%
32	Cloud testing	0.06	\$4.38	\$43,750	0.25%
33	Field (Beta) testing	0.03	\$2.63	\$26,250	0.12%

34	Acceptance testing	0.03	\$2.10	\$21,000	0.12%
35	Independent testing	0.02	\$1.75	\$17,500	0.10%
36	Quality assurance	0.18	\$14.00	\$140,000	0.82%
37	Installation/training	0.03	\$2.63	\$26,250	0.15%
38	Project measurement	0.01	\$1.11	\$11,053	0.06%
39	Project office	0.24	\$19.09	\$190,909	1.11%
40	Project management	1.76	\$140.00	\$1,400,000	8.15%
Cumulative Results		21.91	\$1,743.08	\$17,168,521	100.00%

In table 22 the activities that are related to software quality are highlighted in blue. Out of a total of 40 activities 26 of them are directly related to quality and defect removal. These 26 quality-related activities sum to 50.50% of software development costs while actual coding is only 30.58% of development costs. The accumulated costs for defect-related activities were \$8,670,476. The author is not aware of any other industry where defect-related costs sum to more than half of total development costs. This is due to the high error content of custom designs and manual coding, rather than construction of software from certified reusable components.

So long as software is built using custom designs and manual coding defect detection and defect removal must be the major cost drivers of all software applications. Construction of software from certified reusable components would greatly increase software productivity and benefit the economics of not only software itself but of all industries that depend on software, which essentially means every industry in the world. Table 22 shows the level of granularity needed to understand the cost structures of large software applications where coding is just over 30% of the total effort. Software management and C-level executives such as Chief Financial Officers (CFO) and Chief Information Officers (CIO) need to understand the complete set of activity-based costs and also costs by occupation group such as business analysts and architects over and above programmers.

When you build a house you need to know the costs of everything: foundations, framing, electrical systems, roofing, plumbing etc. You also need to know the separate costs of architects, carpenters, plumbers, electricians, and all of the other occupations that work on the house. Here too for large systems in the 10,000 function point size range a proper understanding of software economics needs measurements of ALL activities and all occupation groups and not just coding programmers, whose effort is often less than 30% of the total effort for large systems.

Both LOC metrics and cost per defect metrics should probably be viewed as *professional malpractice* for software economic studies because they both distort reality and make bad results look better than good results. It is no wonder that software progress resembles and drunkard's walk when hardly anybody knows how to measure either quality or productivity with metrics that make sense and match standard economics.

Software's Lack of Accurate Data and Poor Education on Quality and Cost of Quality (COQ)

One would think that software manufacturing economics would be taught in colleges and universities as part of computer science and software engineering curricula, but universities are essentially silent on the topic of fixed costs probably because the software faculty does not understand software manufacturing economics either. There are a few exceptions such as the University of Montreal however. The private software education companies and the professional associations are also silent on the topic of software economics and the hazards of cost per defect and lines of code. It is doubtful if either of these sectors understands software economics well enough to teach it. They certainly don't seem to understand either function points or quality metrics such as defect removal efficiency (DRE). Even more surprising some of the major software consulting groups with offices and clients all over the world are also silent on software economics and the hazards of both cost per defect and lines of code. Gartner Group uses function points but apparently has not dealt with the impact of fixed costs and the distortions caused by the LOC and cost per defect metrics.

You would think that major software quality tool vendors such as those selling automated test tools, static analysis tools, defect tracking tools, automated correctness proofs, or test-case design methods based on cause-effect graphs or design of experiments would measure defect potentials and DRE because these metrics could help to demonstrate the value of their products. Recall that IBM used defect potentials and DRE metrics to prove the value of formal inspections back in 1973.

But the quality companies are just as clueless as their clients when it comes to defect potentials and defect removal efficiency (DRE) and the economic value of high quality. They make vast claims of quality improvements but provide zero quantitative data. For example only CAST Software that sells static analysis uses function points on a regular basis from among the major quality tool companies. But even CAST does not use defect potentials and DRE although some of their clients do. You would also think that project management tool companies that market tools for progress and cost accumulation reporting and project dashboards would support function points and show useful economic metrics such as work hours per function point and cost per function point. You would also think they would support activity-based costs. However most project management tools do not support either function point metrics or activity-based costs, although a few do support earned value and some forms of activity-based cost analysis. This means that standard project management tools are not useful for software benchmarks since function points are the major benchmark metric.

The only companies and organizations that seem to know how to measure quality and economic productivity are the function point associations such as COSMIC, FISMA, IFPUG, and NESMA; the software benchmark organizations such as ISBSG, David's Consulting, Namcook Analytics, TIMetricas, Q/P Management Group, and several others; and some of the companies that sell parametric estimation tools such as KnowledgePlan, SEER, SLIM, and the author's Software Risk Master (SRM). In fact the author's SRM tool predicts software application size in a total of 23 metrics including all forms of function points plus story points, use-case points, physical and logical code, and a number of others. It even predicts bad metrics such as cost per defect and lines of code primarily to demonstrate to clients why those metrics distort reality. Probably not one reader out of 1000 of this paper has quality and cost measures that are accurate enough to confirm or challenge the data in tables 19, 20, and 21 because software measures and metrics have been fundamentally incompetent for over 50 years. This kind of analysis can't be done with "cost per defect" or "lines of code" because they both distort reality and conceal the economic value of software quality.

However the comparatively few companies and fewer government organizations that do measure software costs and quality well using function points and DRE can confirm the results. The quality pioneers of Joseph Juran, W. Edwards Deming, and Phil Crosby showed that for manufactured products quality is not only free it also saves time and money. The same findings are true for software, only software has lagged all other industries in discovering the economic value of high software quality because software metrics and measures have been so bad that they distorted reality and concealed progress. The combination of function point metrics and defect removal efficiency (DRE) measures can finally prove that high software quality, like the quality of manufactured products, lowers development costs and shortens development schedules. High quality also lowers maintenance costs, reduces the odds of successful cyber-attacks, and improves customer satisfaction levels.

Summary and Conclusions

The combination of *defect potentials* and *defect removal efficiency (DRE)* measures provide software engineering and quality personnel with powerful tools for predicting and measuring all forms of defect prevention and all forms of defect removal.

Function points are the best metric for normalizing software defect potentials because function points are the only metrics that can handle requirements, design, architecture, and other sources of non-code defects. This paper uses IFPUG 4.3 function points. Other forms of function point metric such as COSMIC, FISMA, NESMA, etc. would be similar but not identical to the values shown here.

As of 2016 there is insufficient data on SNAP metrics to show defect potentials and defect removal efficiency. However it is suspected that non-functional requirements contribute to defect potentials in a significant fashion. There is insufficient data in 2016 to judge DRE values against non-functional defects. Note that the author's Software Risk Master (SRM) tool predicts defect potentials and defect removal efficiency (DRE) as standard outputs for all projects estimated.

For additional information on 25 methods of pre-test defect removal and 25 forms of testing, see The Economics of Software Quality, Addison Wesley, 2012 by Capers Jones and Olivier Bonsignour.

References and Readings on Software Quality

Beck, Kent; Test-Driven Development; Addison Wesley, Boston, MA; 2002; ISBN 10: 0321146530; 240 pages.

Black, Rex; Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing; Wiley; 2009; ISBN-10 0470404159; 672 pages.

Chelf, Ben and Jetley, Raoul; *"Diagnosing Medical Device Software Defects Using Static Analysis"*; Coverity Technical Report, San Francisco, CA; 2008.

Chess, Brian and West, Jacob; Secure Programming with Static Analysis; Addison Wesley, Boston, MA; 20007; ISBN 13: 978-0321424778; 624 pages.

Cohen, Lou; Quality Function Deployment – How to Make QFD Work for You; Prentice Hall, Upper Saddle River, NJ; 1995; ISBN 10: 0201633302; 368 pages.

Crosby, Philip B.; Quality is Free; New American Library, Mentor Books, New York, NY; 1979; 270 pages.

Everett, Gerald D. And McLeod, Raymond; Software Testing; John Wiley & Sons, Hoboken, NJ; 2007; ISBN 978-0-471-79371-7; 261 pages.

Festinger, Dr. Leon; A Theory of Cognitive Dissonance; Stanford University Press, 1962.

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>.

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

Hallowell, David L.; *Six Sigma Software Metrics, Part 1.*;
<http://software.isixsigma.com/library/content/03910a.asp>.

International Organization for Standards; ISO 9000 / ISO 14000; <http://www.iso.org/iso/en/iso9000-14000/index.html>.

Jones, Capers; Software Risk Master (SRM) tutorial; Namcook Analytics LLC, Narragansett RI, 2015.

Jones, Capers; Software Defect Origins and Removal Methods; Namcook Analytics LLC; Narragansett RI, 2015.

Jones, Capers; The Mess of Software Metrics; Namcook Analytics LLC, Narragansett RI; 2015.

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, Boston, MA; 2011; ISBN 978-0-13-258220-9; 587 pages.
- Jones, Capers; Software Engineering Best Practices; McGraw Hill, New York; 2010; ISBN 978-0-07-162161-8; 660 pages.
- Jones, Capers; Applied Software Measurement; McGraw Hill, 3rd edition 2008; ISBN 978-0-07-150244-3; 662 pages.
- Jones, Capers; Critical Problems in Software Measurement; Information Systems Management Group, 1993; ISBN 1-56909-000-9; 195 pages.
- Jones, Capers; Software Productivity and Quality Today -- The Worldwide Perspective; Information Systems Management Group, 1993; ISBN -156909-001-7; 200 pages.
- Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.
- Jones, Capers; New Directions in Software Management; Information Systems Management Group; ISBN 1-56909-009-2; 150 pages.
- Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press, Boston, MA; December 1995; 250 pages; ISBN 1-850-32804-8; 292 pages.
- Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.
- Jones, Capers; Estimating Software Costs; 2nd edition; McGraw Hill, New York; 2007; 700 pages..
- Jones, Capers; “The Economics of Object-Oriented Software”; SPR Technical Report; Software Productivity Research, Burlington, MA; April 1997; 22 pages.
- Jones, Capers; “Software Project Management Practices: Failure Versus Success”; Crosstalk, October 2004.
- Jones, Capers; “Software Estimating Methods for Large Projects”; Crosstalk, April 2005.
- 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.
- Land, Susan K; Smith, Douglas B; Walz, John Z; Practical Support for Lean Six Sigma Software Process Definition: Using IEEE Software Engineering Standards; WileyBlackwell; 2008; ISBN 10: 0470170808; 312 pages.
- Mosley, Daniel J.; The Handbook of MIS Application Software Testing; Yourdon Press, Prentice Hall; Englewood Cliffs, NJ; 1993; ISBN 0-13-907007-9; 354 pages.
- Myers, Glenford; The Art of Software Testing; John Wiley & Sons, New York; 1979; ISBN 0-471-04328-1; 177 pages.
- Nandyal; Raghav; Making Sense of Software Quality Assurance; Tata McGraw Hill Publishing, New Delhi, India; 2007; ISBN 0-07-063378-9; 350 pages.

Radice, Ronald A.; High Quality Low Cost Software Inspections; Paradoxicon Publishingl Andover, MA; ISBN 0-9645913-1-6; 2002; 479 pages.

Royce, Walker E.; Software Project Management: A Unified Framework; Addison Wesley Longman, Reading, MA; 1998; ISBN 0-201-30958-0.

Wiegers, Karl E.; Peer Reviews in Software – A Practical Guide; Addison Wesley Longman, Boston, MA; ISBN 0-201-73485-0; 2002; 232 pages.

Abran, A.:

Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers

Wiley IEEE Computer Society Press, 2015 (288 pages), ISBN 978-1-118-95408-9



This book introduces theoretical concepts to explain the fundamentals of the design and evaluation of software estimation models. It provides software professionals with vital information on the best software management software out there.

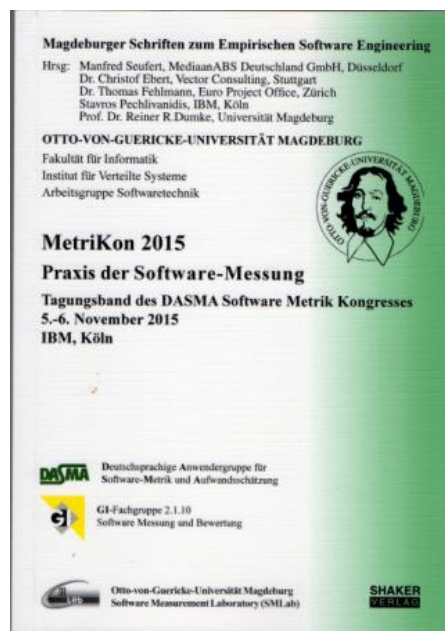
- End-of-chapter exercises
- Over 100 figures illustrating the concepts presented throughout the book
- Examples incorporated with industry data

Seufert, M.; Ebert, C, Fehlmann, T.; Pechlivanidis, S.; Dumke, R. R.:

***MetriKon 2015 - Praxis der Softwaremessung
Tagungsband des DASMA Software Metrik Kongresses
5. - 6. November 2015, IBM, Köln***

Shaker Verlag, Aachen, 2015 (272 Seiten)

The book includes the proceedings of the MetriKon 2015 held in Cologne in November 2015, which constitute a collection of theoretical studies in the field of software measurement and case reports on the application of software metrics in companies and universities.



Schmietendorf, A.; Simon, F.:

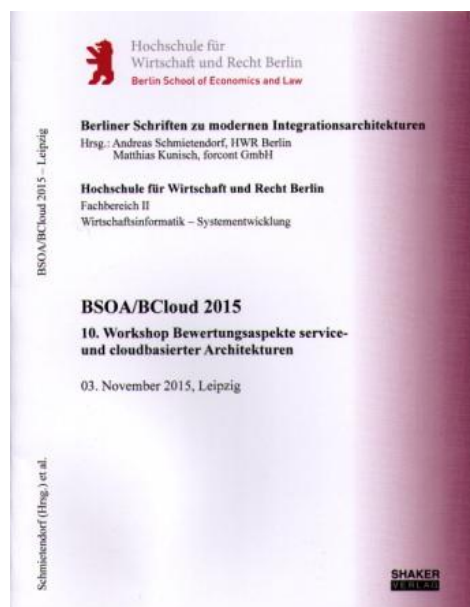
BSOA/BCloud 2015

10. Workshop Bewertungsaspekte serviceorientierter Architekturen

3. November 2015, Leipzig

Shaker Verlag, Aachen, 2015 (112 Seiten), ISBN 978-3-8440-2108-0

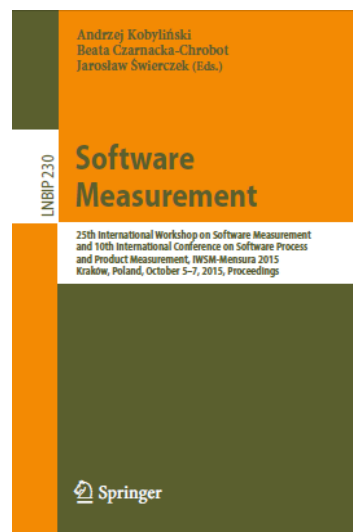
The book includes the proceedings of the BSOA/BCloud 2015 held in Leipzig in November 2015, which constitute a collection of theoretical studies in the field of measurement and evaluation of service oriented and cloud architectures.



Software Measurement

25th International Workshop on Software Measurement and 10th International Conference on Software Process and Product Measurement, IWSM-Mensura 2015
Kraków, Poland, October 5–7, 2015
Proceedings

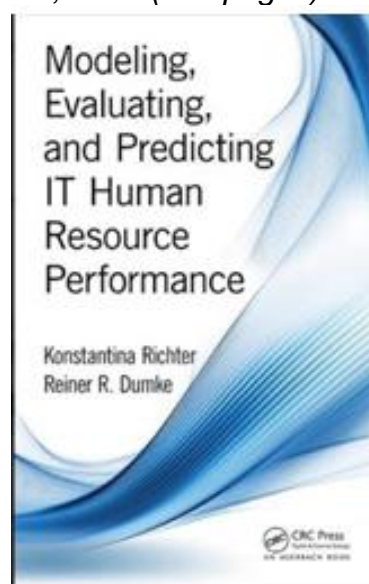
Andrzej Kobyliński · Beata Czarnacka-Chrobot
Jarosław Świerczek (Eds.)



Konstantina Richter, Reiner Dumke:

Modeling, Evaluating and Predicting IT Human Resource Performance

CRC Press, Boca Raton, Florida, 2015 (275 pages)



Schmietendorf, A. (Hrsg.):

***Eine praxisorientierte Bewertung von Architekturen
und Techniken für Big Data***

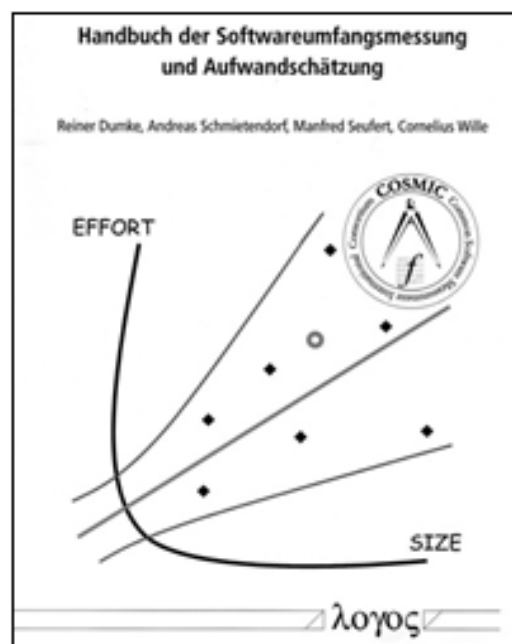
(110 Seiten) Shaker-Verlag Aachen, März 2015 ISBN 978-3-8440-2939-0



Dumke, R., Schmietendorf, A., Seufert, M., Wille, C.:

Handbuch der Softwareumfangsmessung und Aufwandschätzung

Logos Verlag, Berlin, 2014 (570 Seiten), ISBN 978-3-8325-3784-5



Software Measurement & Data Analysis Addressed Conferences

August 2016:

- ICGSE 2016:** **11th International Conference on Global Software Engineering**
August 2 - 5, 2016, Orange Country, California, USA
see: http://www.ics.uci.edu/~icgse2016/2_0cfp.html
- ICSEA 2016:** **10th International Conference on Software Engineering Advances**
August 21 - 25, 2016, Brussels, Belgium
see: <http://www.iaria.org/conferences2016/ICSEA16.html>
- QEST 2016:** **13th International Conference on Quantitative Evaluation of Systems**
August 23 - 25, 2016, Quebec City, Canada
see: <http://www.qest.org/>
- ICDSE 2016:** **International Conference on Data Science and Engineering**
August 23 - 25, Kerala, India
See: <http://icdse.cusat.ac.in/>
- Euromicro SEAA 2016:** **DSD/ Software Engineering & Advanced Application Conference**
August 31 - September 2, 2016, Limassol, Cyprus
see: <http://dsd-seaa2016.cs.ucy.ac.cy/>

September 2016:

- ESEM 2016:** **10th International Symposium on Empirical Software Engineering & Measurement**
September 8 - 9, 2016, Ciudad Real, Spain
see: <http://alarcos.esi.uclm.es/eseiw2016/esem/>
- RE 2016:** **24th IEEE International Requirement Engineering Conference**
September 12 - 16, 2016, Beijing, China
see: <http://re16.org/>
- EuroAsiaSPI² 2016:** **23th European Systems & Software Process Improvement and Innovation Conference,**
September 14 - 16, 2016, Graz, Austria
see: <http://www.eurospi.net/>
- ASQT 2016:** **Arbeitskonferenz Softwarequalität, Test und Innovation**
September 21 - 23, 2016, Klagenfurt, Austria

see: <http://www.asqt.org/>

Big Data 2016: **Big Data Analysis and Data Mining**
September 26 - 27, 2016, London, UK
See: <http://datamining.conferenceseries.com/>

October 2016:

IWSM-MENSURA 2016: **Common International Conference on Software Measurement**
October 5 - 7, 2016, Berlin, Germany
see: <http://www.iwsm-mensura.org/>

ISSRE 2016: **27th International IEEE Symposium on Software Reliability Engineering**
October 23 - 27, 2016, Ottawa, Canada
see: <http://issre.net/>

November 2016:

BSOA/BCloud 2016: **11. Workshop Bewertungsaspekte service-orientierte und Cloud-Architekturen**
November , 2016, Berlin, Germany
see: <http://www-ivs.cs.uni-magdeburg.de/~gi-bsoa/>

ICDM 2016: **IEEE International Conference on Data Mining**
November 28 - 30, 2016, Barcelona, Spain
See: <http://icdm2016.eurecat.org/>

December 2016:

PROFES 2015: **16th International Conference on Product Focused Software Process Improvement**
December 2 - 4, 2015, Bolzano, Italy
see: <http://profes2015.inf.unibz.it/> **(not in 2016)**

see also: Conferences Link of **Luigi Buglione** (<http://www.semq.eu/leng/eveprospi.htm>)

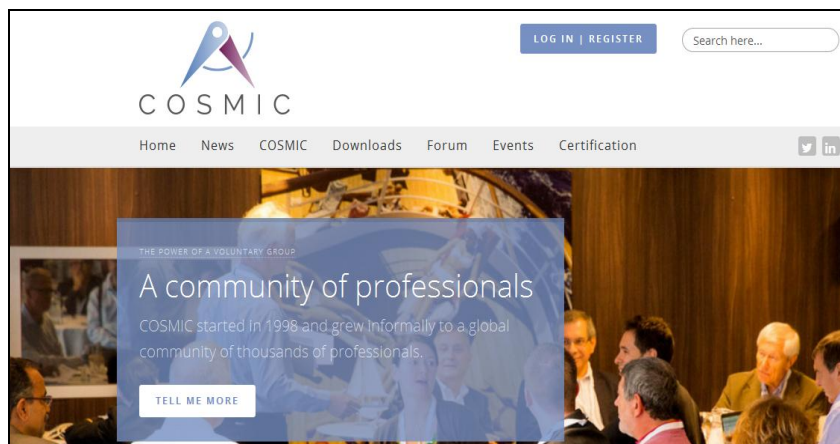
See the GI-Web site <http://fg-metriken.gi.de/> for the digital contents of the Software Measurement News:



Help to qualify the software measurement knowledge and intentions in the world wide web:



cosmic-sizing.org:



See our overview about software metrics and measurement in the Bibliografie at <http://fg-metriken.gi.de/bibliografie.html> including any hundreds of books and papers:

Gesellschaft für Informatik

Fachgruppe Software-Messung und -Bewertung

Startseite | Vorstand | Aktuelles | **Bibliografie** | Arbeitskreise | Software Measurement News

Sie befinden sich hier: Startseite/Bibliografie

Software Measurement Bibliography

Basisliteratur finden Sie hier

1 Software Measurement Foundations

- Measurement Overview
- Measurement Principles & Foundations
- Measurement Standards
- Basic (Set of) Measures
- Measurement Validation
- Measurement & Statistics

2 Software Process & Product Measurement

2.1 Software Product Measurement

- Requirements Measurement
- Review/Inspection/Audits Measurements
- Specification Measurement
- Architecture Measurement
- Product Line Measurement
- Reliability Measurement
- Security Measurement
- Performance Measurement

2.2 Software Process & Resources Measurement

- Process Management & Measurements
- Risk Management & Measurements
- Process Improvement Measurements
- Quality Assurance Measurements
- Software Test Measurements
- Maintenance Measurements
- Personal Measurements
- SPC (Statistical Process Control)

2.3 Software Process Maturity Models and Measurement

- General Maturity Quantifications
- CMMI/SPICE/ITIL Measures
- Maintenance Maturity Measures (S3M)
- Agile Process Measures (AMMI)

2.4 Software Size Measurement & Cost/Effort Estimation

- Size Measurement
- Cost/Effort Estimation
- COCOMO (Constructive Cost Model)
- (Empirical-based) Function Points
- Use Case Points
- Object/Data/Feature Points

2.5 Software Metrics Tools and Infrastructures

- Program (Source Code) Measurement Tools
- Metric Analysis Tools
- Process Management & Measurement Tools
- SQA (Software Quality Assurance) Tools
- Metrics Data Bases & Experience Factories
- Software Benchmarks & ISBSG (International Software Benchmark Standard Group)
- Cockpits/Dashboards & Measurement Infrastructures
- Measurement Services, eMeasurement & WEB Tomography

3 Measurement of Software Paradigms and Technologies

3.1 Procedural-Based Software Engineering (PBSE)

- Source Code Measurements (Halstead, McCabe etc.)
- Control/Data Flow Measures
- Module/Procedure Measures

3.2 Object-Oriented Software Engineering (OOSE)

- Object Oriented Design Measures
- Object Oriented Programming Measures
- Object Oriented Test Measures
- Aspect-Oriented Measures

3.3 Applicative & Declarative Approaches (Calculus)

- Functional Programming Measurement
- Logical Programming Measurement
- Formal Specification Measurement

3.4 Component-Based Software Engineering (CBSE)

- Components (COTS) Measurements
- EAI Measurements
- Reusability Measurement
- Feature-Oriented Measurement

3.5 Service-Oriented Software Engineering (SOSE)

- Web (Service) Measurement
- SOA (Service-Oriented Architecture) Measurement
- Cloud Measurement
- Big Data Measurement

3.6 Agent-Oriented Software Engineering (AOSE)

- Software Agents Measurement
- MAS (Multi Agent System) Measurement
- MAS Development Measurement
- Self Management & Measurement

4 Measurement Frameworks

4.1 General Measurement Approaches

- General Frameworks
- GQM (Goal Question Metric)
- PDCA (Plan, Do Check, Act)
- E4 (Establish, Extract, Evaluate, Execute)
- CAME (Choice, Adjust, Migration, Efficiency)
- DMIAC (Define, Measure, Analyze, Improve, Control)

4.2 Modern Measurement Frameworks

- Adaptive Measurement Frameworks
- Proactive Measurement Frameworks
- Ontology-Based Measurement Frameworks
- Categorical Theory Based Approaches
- Ubiquitous Measurement Frameworks

4.3 Measurement Process Evaluations

- Measurement Evidence & Evaluation
- Measurement Process Classification

See our further software measurement and related communities:

www.dasma.org:

Deutschesprachige Anwendergruppe für Software-Metrik und Aufwandschätzung e.V.

Welcoming to DASMA!

AKTUELLES

- Metrikon 2015 - Praxis der Softwaremessung**
Die von der Dasma und der DSI organisierte und in Deutschland einzigartige Fachtagung Metrikon findet bei IBM in Köln am 5. + 6. November statt. Informationen zum Call for Paper und zu den Keynotes unter www.metrikon.de
- DASMA Zukunftspreis 2015**
Bereits zum 13. Mal wird der mit 2000 € dotierte Zukunftspreis für herausragende Studienarbeiten von der DASMA verliehen. [Mehr Informationen](#) zu den Themengebieten und dem Prozess zur Einreichung 2015.
- IWSH-MENSURA 2015 CFP**
Kirkau - die wunderschöne polnische Residenzstadt an der Weichsel wird vom 5. bis zum 7. Oktober 2015 die IWSH (Warsaw) beherbergen. Neben einem hochinteressanten wissenschaftlichen Programm mit Vorträgen, Workshops und Tutorien bietet Kirkau viele Sehenswürdigkeiten, unter anderem die berühmten Backsteinen mit ihren wehrartigen Höhen. Kirkau ist mit Auto, Fahrrad, Zug und Flugzeug sehr gut erreichbar. Wir freuen uns auf eine große Beteiligung und Ihre Beiträge. Diese sollten bis zum 3. Mai 2015 angemeldet werden. Details finden Sie [hier](#).
- Der 10. BSOA-Workshop in Leipzig am 03. November 2015**
Beratungsaspekte service- und cloud-as-a-entwurf (Architecture) API economy: From SOA to ICA. DASMA-Mitglieder profitieren wie jedes Jahr von einer ermäßigten Teilnahmegebühr. [Mehr Informationen](#)

DASMA - Ihre erste Adresse rund um Software-Metriken und

www.isbsg.org:

ISBSG | The global and independent source of data and analysis for the IT industry

Use industry history data to improve your IT management
Click on your area of interest below:

- Software Development & Enhancement**
Improve IT performance through estimation, benchmarking, project planning & management and IT infrastructure planning
- Software Maintenance & Support**
Improve management performance of software portfolio maintenance and support

Case Studies
Large Bank, Netherlands
The ISBSG data helped this organization to estimate their project in a realistic way, preventing them from a huge failure. [View case study](#)

News
New Report: Performance by Country
3 April 2015
Using the 5,000 projects in the repository we take a look at the characteristics of software.

The ISBSG thanks the following supporting organisations:

SEER - GATORATH

www.cecmg.de:

ceCMG | Central European Computer Measurement Group

Startseite | Veranstaltungen | ceCMG | Information | Downloads

Wichtige Hinweise
06-07 2013 März Jahresstimmung

Vorträge zur Jahrestagung 2013 in Gelsenkirchen
In unserem Download-Bereich finden Sie eine Übersicht über die Vorträge, die auf der Jahrestagung 2013 in Gelsenkirchen stattfinden werden. Darüber hinaus stellen wir unsere Speaker kurz vor.

Einladung zum Workshop: Performance Management Update for z/OS System Programmers
Liebe Mitglieder und Interessenten der ceCMG
Aufgrund einer besonders und einmaligen Gelegenheit können wir Ihnen dieses Jahr in Vorfeld der Jahrestagung 2013 einen ganz

Aussteller/Sponsoren der Jahrestagung 2013:
25 Jahre
Aussteller
K
Intercom
Computer
Systeme
Aussteller
NextStride

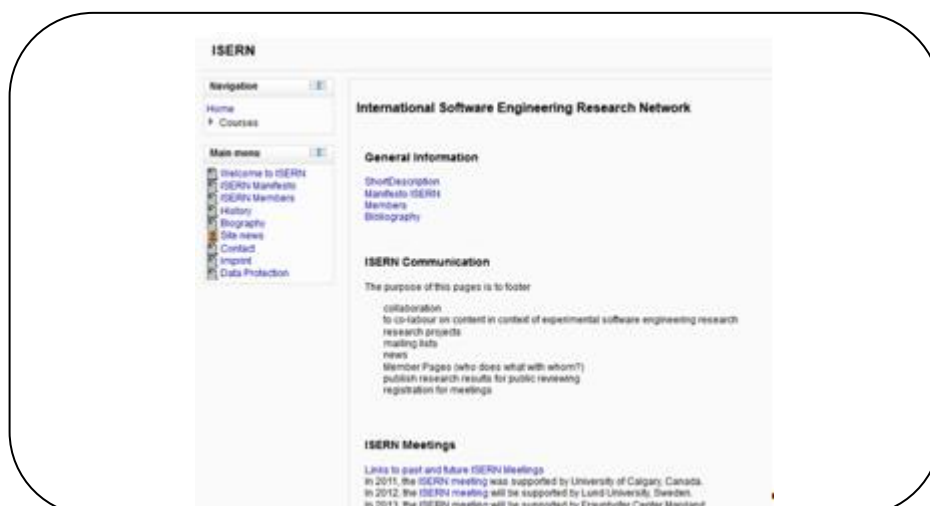
www.mai-net.org:



www.swebok.org:



isern.iese.de:




www.smlab.de:



www.psm-sc.com/:



sebokwiki.org/wiki/Measurement:



SEBoK
Guide to the Systems Engineering Body of Knowledge

Page Read View source Go Search

Measurement

Measurement

Measurement and the accompanying analysis are fundamental elements of **systems engineering** (SE) and technical management. SE measurement provides information relating to the products developed, services provided, and processes implemented to support effective management of the processes and to objectively evaluate product or service quality. Measurement supports realistic planning, provides insight into actual performance, and facilitates assessment of suitable actions (Roedler and Jones 2005, 1-65; Frenz et al. 2010).

Appropriate measures and indicators are essential inputs to tradeoff analyses to balance cost, schedule, and technical objectives. Periodic analysis of the relationships between measurement results and review of the requirements and attributes of the system provides insights that help to identify issues early, when they can be resolved with less impact. Historical data, together with project or organizational context information, forms the basis for the predictive models and methods that should be used.

Contents [hide]

1 Fundamental Concepts

www.fisma.fi/in-english/:



FISMA
Finnish Software Measurement Association

SPN Perustiedot Toiminta Kalereri Uutiset Yhteystiedot In English

Scope Management
Activities
SPN
Methods
Contact info

Turku Järvenpää
Seinäjoki
Akaa

In English

FISMA – For better Management

Finnish Software Measurement Association FISMA is a non-profit, independent association focusing on better management through improving the quality and measurability of software & systems engineering and IT service management.

FISMA's membership is intended for all companies, research units, universities and other institutes interested in software measurement. At the moment, there are about 40 active member organisations and local software process improvement networks (SPNis).

FISMA is actively participating standardisation activities and perform as a national body of Finland in developing standards under ISO/IEC JTC1 SC7, Software and Systems Engineering subcommittee.

Areas of standardisation which FISMA follows particularly:

- Software and systems engineering frameworks (ISO/IEC 12267, ISO/IEC 15268, ISO/IEC 15504, CIM0)
- Measurement of software projects (ISO/IEC 14143, ISO/IEC 29081)

Kalereri

Research Forum 1_2015
3.3.2015 klo 10:15-12:15

Scope Manager Forum
kokous
10.3.2015 klo 13-18

22nd EuroAsiaSPR
(EuroSPR) Conference
2015, call for papers
25.5.2015

[Lisää tapahtumia](#)

<http://nesma.org/>:



www.sei.cmu.edu/measurement/:



<http://www.omg.org/news/releases/pr2013/02-07-13.htm>:

CORBA		CWM		OMG		BPMN		UML		MARTE	
V-F		ML		DDS		UML		MARTE		MODEL	
About Us		Press Room		Calendar		Documents		Members Only		Technology	
Industries		Programs									

Contact:
 Julie Pike
 OMG
 +1-781-444 0404
julie@omg.org

Cloud Standards Customer Council Hosts "Big Data in the Cloud" Conference at Meeting in Reston, VA
 Will Host Webinar on February 14th to Introduce Conference

Needham, MA- 02-07-2013- The Cloud Standards Customer Council (CSCC) will be hosting the "Big Data in the Cloud: Preparing for the Future" conference

SOFTWARE MEASUREMENT NEWS

VOLUME 21

2016

NUMBER 2

CONTENTS

Announcements 3

Position Paper 9

Christof Ebert

Cyclomatic Complexity - 40 Years Later 9

Capers Jones

The Origins of Function Point Metrics 12

Andreas Schmietendorf

Web APIs als Enabler einer erfolgreichen Digitalisierungsstrategie 15

Capers Jones

Exceeding 99% in Defect Removal Efficiency (DRE) for Software 19

New Books on Software Measurement 45

Conferences Addressing Measurement Issues 49

Metrics in the World-Wide Web 51

ISSN 1867-9196