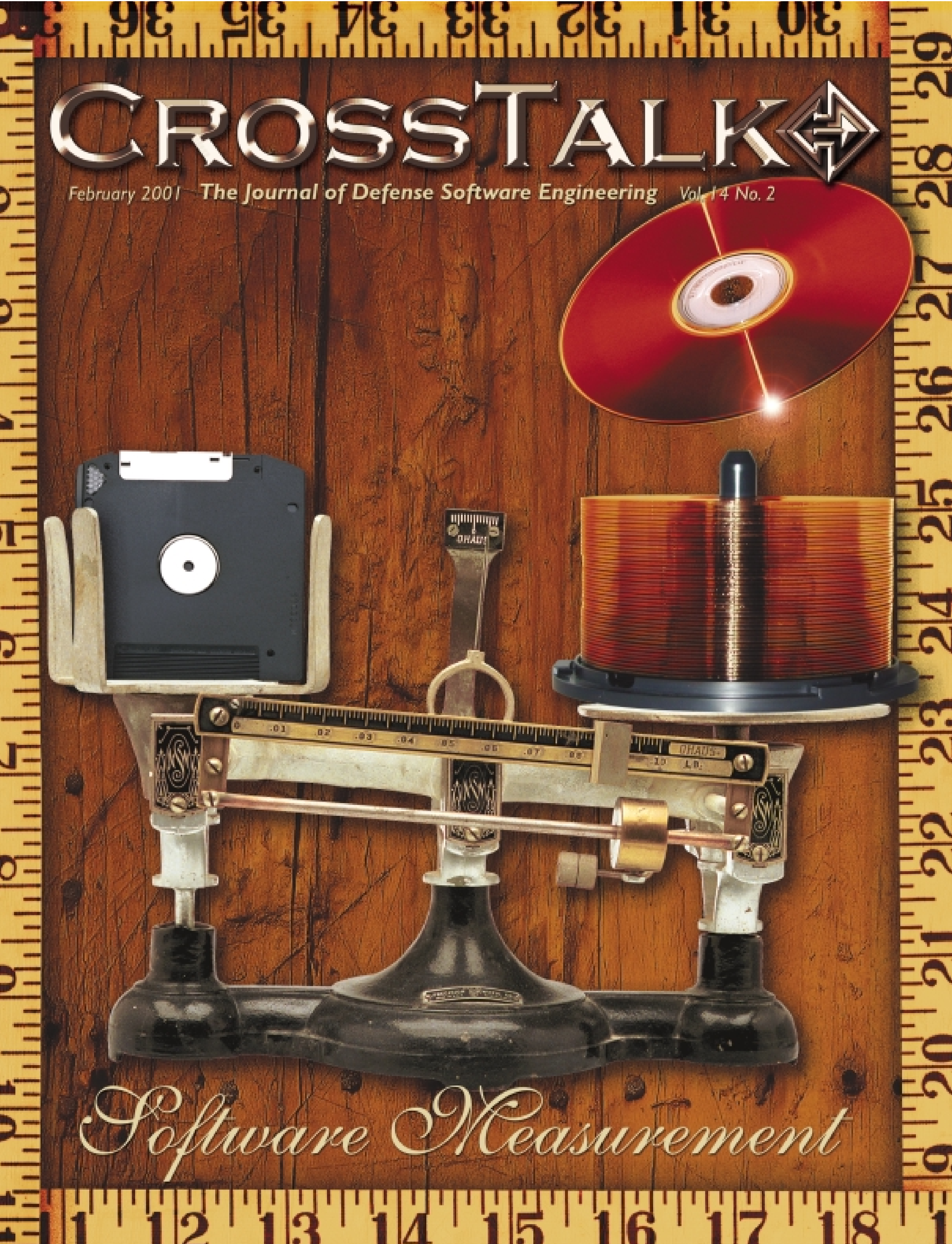

CrossTalk
February 2001
The Journal of Defense Software Engineering
Vol. 14 No. 2
Software Measurement

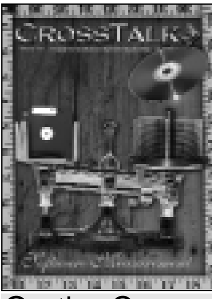
Software Measurement

4 Software Measurement Programs and Industry Leadership
A review of consulting studies shows that companies that are successfully improving quality and schedules are the ones with the best measurements.
by Capers Jones

8 Applying Function Point Analysis to Requirements Completeness
Function point analysis provides analysts with one extra frame of reference to gauge completeness of known user requirements.
by Carol Dekkers and Mauricio Aguiar

12 Measure Size, Complexity of Algorithms Using Function Points
Breaking down algorithms into functional components is a repeatable and reliable way to measure size and complexity, estimate cost, and develop schedule.
by Nancy Redgate and Charles B. Tichenor

16 The Nine-Step Metrics Program
The Software Technology Support Center at Hill AFB, Utah, developed a process that logically groups the SEI's CMMI measurement steps by activity type.
by Timothy K. Perkins



On the Cover: Kent Bingham, Digital Illustration and Design, is a self-taught graphic artist/designer and freelances print and Web design projects. His portfolio is at www.adobe.com/eportfolio/kentbingham

Best Practices

20 CMM Level 4 Quantitative Analysis and Defect Prevention
Statistical process control can provide valuable information that is used in defect prevention and lessons learned during software development.
by Al Florence

Open Forum

24 Evolving Function Points
Findings show that simple semantic changes eliminate inconsistencies and make function points easier to learn.
by Lee Fischman

Departments

3 From the Publisher

7 Letter to the Editor

19 Measurement Web Sites

27 Quote Marks

27 Coming Events

28 STC 2001 Announcement

31 BackTalk

CrossTalk

SPONSOR *Lt. Col. Glenn A. Palmer*

PUBLISHER *Reuel S. Alder*

ASSOCIATE PUBLISHER *Elizabeth Starrett*

MANAGING EDITOR *Pam Bowers*

ASSOCIATE EDITOR/LAYOUT *Matthew Welker*

ASSOCIATE EDITOR/FEATURES *Heather Winward*

GRAPHIC DESIGNER *Abby Hall*

VOICE 801-586-0095
FAX 801-777-5633
E-MAIL crosstalk.staff@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP ONLINE www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 30.

Ogden ALC/TISE
5851 F Ave., Bldg 849, Rm B-04
Hill AFB, Utah 84056-5713

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at www.stsc.hill.af.mil/CrossTalk/xtlkguid.pdf. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: at www.stsc.hill.af.mil.
Call 801-777-7026, e-mail: randy.schreifels@hill.af.mil.

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Software Measurement Is Inherent to Project Success

My belief and dedication in the notion that institutionalized measurement is a critical factor in the success of software projects goes back many years. As a quality manager for a state-of-the-art computer-aided engineering software system, I verified the needs and benefits of measuring and analyzing software product quality more than 15 years ago. Since then I have spent many years in the promotion and improvement of measurement as a basic tool in software and systems development. This issue of CrossTalk aims to give our readers a few ideas regarding using *measurement* in a software environment.

In *The Nine-Step Metrics Program,* Tim Perkins covers the basics in considering how to set up a measurement program. He provides some insight for beginners on what measurements to collect and how to use measurements once they are collected.

At the other end of the maturity scale, Al Florence's article *CMM Level 4 Quantitative Analysis and Defect Prevention* gives project examples of how the use of rigorous statistics can easily and effectively be used in a software setting. His real-project examples detail how focusing on product quality allows a high maturity organization to move forward in defect prevention.

Lee Fischman's article *Evolving Function Points* analyzes the question: "What is wrong with function points?" He recommends a few changes to current function point methods that would result in a more user-friendly function point standard.

This issue also contains two articles focusing on somewhat unique uses of function points. The first relates to using function points to help measure size and complexity of software algorithms and is written by Nancy Redgate and Charles B. Tichenor. They describe a method that breaks down a mathematical algorithm into its functional components to produce a repeatable and reliable method for determining algorithm size and complexity. The second article is *Applying Function Point Analysis to Requirements Completeness* by Carol Dekkers and Mauricio Aguiar. As the title indicates, this article highlights how the software sizing technique "function point analysis" can be a valuable tool and a structured method for doing a requirements review.

In *Software Measurement Programs and Industry Leadership,* Capers Jones significantly points out that the most successful companies are those with very sophisticated quality and productivity measurement programs in place. He reviews the measures used by leading businesses showing how these are the companies most successful in improving quality and shortening delivery schedules.

Readers should also be aware that a new version of Practical Software and Systems Measurement is being released. The effort has now evolved into an updated version encompassing systems engineering measurements as well as software measurements. Check the Web site at www.psmsc.com for this latest version and information on PSM's Technical Working Group meeting scheduled Feb. 13-14, 2001 in Herndon, Va.

I hope this issue of CrossTalk along with other resources pointed to within this issue will provide the ability to move forward in improving your organization's understanding and use of measurement.

H. Bruce Allgood
Section Chief, TISEA

Software Measurement Programs and Industry Leadership

Capers Jones
Software Productivitiy Research Inc.

Consulting studies carried out by the author and his colleagues at several hundred companies and some government agencies shows that the leading organizations tend to have very sophisticated quality and productivity measurement programs in place. This article covers the highlights of the measurement programs noted among companies in the top 25 percent of productivity and quality results in the United States.

This author and Software Productivity Research Inc. were able to gather data on software projects from more than 600 companies and about 25 government agencies. The total volume accumulated since 1985 amounts to more than 10,000 software projects. Most of our clients are large Fortune 500 corporations although we have done studies for several defense contractors, as well as for the Air Force and Navy.

This article summarizes the best measurement practices noted among these clients, which are in the upper quartile of overall productivity measured in terms of function points per staff month. Unfortunately there are no military or defense projects in the upper 25 percent of productivity data, or even in the upper half. This is due to older military standards such as the Department of Defense (DoD) 2167, which triggered the production of specifications and control documents nearly three times larger than civilian norms for the same size projects. Although many military projects are in the upper quartile in terms of software quality, the DoD is on the bottom in terms of productivity levels. Therefore most of the measurement practices described here were noted among large Fortune 500 companies. However, we have observed similar sets of measurements among top defense contractors.

In general, software is a troubled technology plagued by project failures, cost overruns, schedule overruns, and poor quality levels. Even major corporations such as Microsoft have trouble meeting published commitments, or shipping trouble-free software. But not all companies experience software disasters. Some have mastered software development and can achieve better than average results. When assessing successful software development companies, we always encounter sophisticated measurement programs.

In every industry there are significant differences between the leaders and the laggards in terms of market shares, technological sophistication, and quality and productivity levels. In the software industry one of the most significant differences is that the leaders know their quality and productivity levels because they measure them. The laggards do not measure. Therefore the laggards do not have a clue as to how good or bad they are. Consider three basic comparisons between your company and its competitors:

- Is your software quality better?
- Is your software productivity better?
- Is your company's time to market better?

If you cannot answer these questions, what do you think are your chances to compete against enterprises that do know the answers? If you do not know your software quality and productivity rates, your company and your job are at risk. Your company may also face litigation risk.

Measurement is only part of a suite of key factors leading to software excellence that includes:

- Good measurements.
- Good managers and technical staffs.
- Good development and maintenance processes.
- Complete software tool suites for managers and developers.
- Good organization structures.
- Specialized staff skills.
- On-the-job training.
- Good personnel policies.
- Good office environments.
- Good communications.

But measurement is a root technology that allows companies to make visible progress in improving the other factors. Without good measurements, progress is slow and may even turn negative.

Companies that do not measure tend to waste scarce investment dollars on approaches that consume time and energy but accomplish very little. Surprisingly, investment in good quality and productivity measurement programs has one of the best returns on investment of any known software technology.

Qualities that Industry Leaders Measure

The best way to decide what to measure is to find out what industry leaders measure, and then measure the same things. Following are the kinds of measurements we have noted at companies that have achieved or exceeded Level 3 on the Software Engineering Institute's (SEI) Capability Maturity Model®, have won Malcolm Baldrige awards, and are widely respected within the industry. These same companies were in the top quartile for all companies in terms of software quality and productivity levels.

Every leading company measures software quality. There are no exceptions. If a company does not measure quality, it is not an industry leader, and there is a good chance that software quality levels are hazardous. Quality is the most important topic of measurement. Here are the most important quality measures:

Customer Satisfaction

All leaders perform annual or semiannual customer satisfaction surveys. They also provide blank defect reports or customer complaint forms as part of the user manuals or inside software packaging. Defect reports via the Internet are also supported by industry leaders. Many leaders in the commercial software world have very active user groups and forums on information services. These groups often produce independent quality and satisfaction surveys.

® The Capability Maturity Model and CMM are registered trademarks of the Software Engineering Institute and Carnegie Mellon University.

Defect Quantities
Leaders keep accurate records of bugs or defects found in all major deliverables; they start early during requirements or design. At least five categories of defects are measured:
1. Requirements defects.
2. Design defects.
3. Code defects.
4. Documentation defects.
5. Bad fixes (secondary bugs introduced accidentally while fixing another bug).

Defect Removal
Leaders know the average and maximum efficiency of every major type of review, inspection, and test; they select optimum series of removal steps for projects of various kinds and sizes. Pretest reviews and inspections are normal among organizations with ultra-high quality, since testing alone is not efficient enough. Leaders remove from 95 percent to more than 99 percent of all defects prior to software delivery. Laggards seldom exceed 80 percent defect removal efficiency, and may drop below 50 percent.

Delivered Defects
Leaders begin to accumulate user-reported error statistics as soon as the software is delivered. Monthly reports are prepared and given to executives to show the defect trends against all products. These reports are summarized on an annual basis. Supplemental statistics such as defect reports by country, state, industry, client, etc. are also included.

Defect Severities
All industry leaders, without exception, use some type of severity scale to evaluate incoming bugs or defects reported from the field. The number of plateaus vary from one to five. In general, "Severity 1" are problems that cause the system to fail completely, then the severity scale descends in seriousness.

Reliability/Availability
Application reliability is normally measured using mean time between failures, or for new products mean time to failure. These measures are easier to gather for in-house applications than for commercial software since failure reports are indirect if external customers are involved. Availability is another aspect of reliability, i.e. the percentage of normal work periods the application can be used as intended. Availability is normally measured as a percentage of the work period during which the application is ready and can be run successfully. Anything under about 99 percent tends to generate dissatisfaction.

Service Response Time
This is a measure of how long it takes the software maintenance group to perform key activities such as acknowledge an incoming bug report, repair the bug, and get the fix back out to the user. Another key metric for commercial software is how long it takes a customer to report a bug to a live person. Being put on hold for more than two minutes is a bad sign. Best-in-class organizations are good in all of these factors and can turn around high-severity bugs in 24 hours or less.

Complexity
It has been known for many years that complex code is difficult to maintain and has higher than average defect rates. A variety of complexity analysis tools are commercially available that support standard complexity measures such as cyclomatic and essential complexity.

Test Coverage
Software testing may or may not cover every branch and pathway through applications. A variety of commercial tools are available that monitor the results of software testing, and help identify portions of applications where testing is sparse or nonexistent.

Cost of Quality
The basic cost-of-quality concept originated with Philip Crosby when he worked at ITT [1]. It was aimed at manufacturing and was not a perfect software fit. Therfore most major software shops have modified the original Crosby concepts when measuring. One significant aspect of quality measurement is to keep accurate records of the costs and resources associated with various forms of defect prevention and removal. For software these measures include the following costs:
- Software assessments.
- Quality baseline studies.
- Reviews, inspections, and testing.
- Warranty repairs and post-release maintenance.
- Quality tools; the costs of quality education.
- Your software quality assurance organization.
- User satisfaction surveys.
- Any litigation involving poor quality or customer losses attributed to poor quality.

Productivity and Schedule Measures
Measuring software schedules, effort, and costs are important areas that differentiate leaders from laggards. Many software leaders have also adopted function point metrics rather than the flawed *lines of code* metric. Almost all of the published benchmarks for software are expressed in terms of function points, so there are no viable alternatives for comparative studies. Here are the key productivity-related measures of leading software producers:

Size Measures
Industry leaders measure the sizes of major deliverables associated with software projects. Size data are kept in two ways. One method records the sizes of actual deliverables such as pages of specifications, pages of user manuals, screens, test cases, and volumes of source code. The second method normalizes the data for comparative purposes. Here function point metrics are the most common. Examples would be pages of specifications produced per function point, source code produced per function point, and test cases produced per function point.

Function point metrics were developed by IBM and put into the public domain in 1978. The rules for counting function point metrics are defined by the nonprofit International Function Point Users Group (IFPUG).[1]

Activity-Based Schedule Measures
Leading companies measure the schedules of every activity, and how those activities overlap or are carried out in parallel. Laggards, if they measure schedules at all, simply measure the gross

schedule from the rough beginning of a project to delivery, without any fine structure. Gross schedule measurements are totally inadequate for any kind of serious process improvement analysis.

Activity-Based Cost Measures

Leaders measure the effort for every activity, from requirements to maintenance. When measuring technical effort, leaders measure all activities, including requirements, design, coding, technical documentation, integration, quality assurance, etc. Leaders tend to have a rather complete chart of accounts with no serious gaps or omissions. Laggards either do not measure at all, or collect only project-level data that is inadequate for serious economic studies.

Three kinds of normalized data are typically created:

- Work hours per function point by activity and in total.
- Function points produced per staff month by activity and in total.
- Cost per function point by activity and in total.

Costs are the most subtle and difficult of the productivity-related measures because of large variances in salaries and even larger variances in burden or overhead rates.

Indirect Cost Measures

Leading companies measure costs of direct and indirect activities. Some of the indirect activities such as travel, meeting costs, moving, living, and legal expenses are so costly that they cannot be overlooked.

Monthly Milestone Reports

Leading companies are very good in monitoring progress and normally track every large and important project on a monthly basis. Monthly status reports include cost variance reports, milestone reports, and red-flag items, which are defined as situations that might delay the project or cause an overrun. Examples of red-flag items might be loss of key personnel, a sudden change in requirements, or some other major issue. These monthly reports are usually five to 10 pages. For large systems, first-line managers create the lowest level reports, and their work is summarized upwards. There are also reports on the completion of major milestones such as internal design, coding, and inspections.

Annual Software Reports

One of the surest signs that an organization has reached *best-in-class* status is when they produce an annual report on software demographics, productivity, quality, assessment results, and other key factors. These reports are usually produced on the same cycle as corporate annual reports (i.e. within 90 days from the close of the prior business year). Because software is usually one of the most expensive and labor-intensive commodities in history, it is appropriate to create an annual report for top corporate executives. The annual software reports for a Fortune 500 company are about 60 to 75 pages with sections for each major business unit, industry trends, technology trends, and quantitative and qualitative results.

Assessment Measurements

Even accurate quality and productivity data are of limited value unless they can explain why some projects are better or worse than others. The influential factors that affect the outcomes of software projects are normally collected by software assessments such as those performed by the SEI, Software Productivity Research, Howard Rubin Associates, Quantitative Software Management, or other consulting companies. In general, assessments cover the following topics:

- **Software Processes.** This deals with the activities from early requirements through deployment. Topics include how the project is designed, what quality assurance steps are used, and how configuration control is managed.
- **Software Tools.** There are more than 3,500 software development tools on the commercial market, and companies have built at least the same number of proprietary tools. It is considerably important to explore the usefulness of these available tools. Thoughtful companies identify gaps and missing features, and use this kind of data for planning improvements.
- **Software Infrastructure.** The number, size, and kinds of departments within large organizations are important topics, as are types of communication across organizational boundaries. Whether a project uses matrix or hierarchical management, and whether or not a project involves multiple cities or countries, exerts a significant impact on results.
- **Software Skills.** Large corporations can have more than 100 different occupation groups within their software domains. Some of these specialists include quality assurance, technical writing, testing, integration and configuration control, network specialists, and many more.
- **Staff and Management Training.** Software personnel, like medical doctors and attorneys, need continuing education to stay current. Leading companies tend to provide from 10 to 15 days of education per year for both technical staff members and software management. Assessments explore this topic.
- **Environment.** Physical office layout and noise levels exert a surprisingly strong influence on software results. The best-in-class organizations typically have fairly good office layouts, while laggards tend to use crowded cubicles or densely packed open offices. Recent additions to environmental measures include telecommuting factors and Web access. Some companies provide home computing facilities and portable computers, too.

Business and Corporate Measures

To this point, measurement has mainly been discussed at the level of individual software projects. There are also important measurements at the corporate level. Here are a few samples of corporate measurements noted among our leading clients:

Salary and Benefit Measures

Many companies perform annual benchmark studies of staff compensation and benefit levels. These are not software studies *per se*, but are carried out in support of the entire corporation.

Portfolio Measures

Major corporations can own from 250,000 to more than 1 million function points of software apportioned across thousands of programs and dozens of systems. Leading enterprises know their portfolio's size, their growth rate, replacement cost, quality levels, and many other factors. This information is important for mergers and acquisitions. It has also been a factor in tax litigation, such as ascertaining the value of the software assets when General Motors acquired Electronic Data Systems.

Market Share Measures

The industry and global leaders know a lot more about their markets, market shares, and competitors than the laggards. For example, industry leaders in the commercial software domain tend to know how every one of their products is selling in every country, and how well competitive products are selling globally.

Competitive Measures

Few companies lack competitors. Industry leaders know much about their competitors' products, market shares, and other important topics. A lot of this kind of information is available from various industry sources such as Dun & Bradstreet, Mead Data Central, *Fortune* and other journals, and from industry studies produced by organizations such as Auerbach, the Gartner Group, and others.

Summary and Conclusions

The software industry is struggling to overcome a very bad reputation for poor quality and long schedules. The companies that have been most successful in improving quality and shortening schedules have also been the ones with the best measurements.

The U.S. software industry is about to face major challenges from overseas vendors with markedly lower labor costs than U.S. norms. Measurement of software quality and productivity is already an important business tool. As off-shore software vendors use metrics and measurements to attract U.S. clients, good measurements may well become a business weapon.◆

References

1. Crosby, Philip B., *Quality is Free*, Mentor Book, New York, N.Y., 1979.

Note

1. Readers wanting more information about function point metrics can access the IFPUG Web site at www.IFPUG.org

About the Author

Capers Jones is chief scientist of Artemis Management Systems and director of Software Productivity Research Inc., Burlington, Mass. Jones is an international consultant on software management topics, a speaker, a seminar leader and author. He is also well known for his company's research programs into critical software issues:

- Software Quality: Survey of the State of the Art.
- Software Process Improvement: Survey of the State of the Art.
- Software Project Management: Suvey of the State of the Art.

Formerly, Jones was assistant director of programming technology at the ITT Programming Technology Center in Stratford, Conn. Prior to that, he was at IBM for 12 years. He received the IBM General Product Division's outstanding contribution award for his work in software quality and productivity improvement methods.

Software Productivity Research Inc.
6 Lincoln Knoll Drive
Burlington, Mass. 01803
Phone: 781-273-0140
Fax: 781-273-5176
E-mail: CJones@SPR.com
Internet: www.spr.com

Additional Readings

- Austin, Robert D., *Measuring and Managing Performance in Organizations*, Dorset House, New York, N.Y., 1996.
- Boehm, Barry W. *Software Engineering Economics* Prentice Hall, Englewood Cliffs, N.J., 1981.
- DeMarco, Tom, *Controlling Software Projects*, Yourdon Press (Prentice Hall), Englewood Cliffs, N.J., 1982.
- Dreger, J. Brian, *Function Point Analysis*, Prentice Hall, Englewood Cliffs, N.J., 1989.
- Grady, Robert B. and Caswell, Deborah L., *Software Metrics: Establishing a Company Wide Program*, Prentice-Hall Inc., 1987.
- Humphrey, Watts, *Managing the Software Process*, Addison-Wesley, Winthrop, Mass., 1989.
- Jones, Capers, *Applied Software Measurement*, McGraw-Hill, New York, N.Y., 1996.
- Jones, Capers, *Software Assessments, Benchmarks, and Best Practices*, Addison Wesley Longman, Boston, Mass., 2000.
- Kan, Stephen H., *Metrics and Models in Software Quality Engineering*, Addison Wesley Longman; Reading, Mass., 1995.
- Paulk, M.C., Curtis, B., Chrissis, Mary Beth, et al, *Capability Model for Software,* Software Engineering Institute, Pittsburgh, Pa., August 1991, SEI Technical Report 24.
- Putnam, Larry, *Measures for Excellence*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- Rubin, Dr. Howard, *Annual Software Benchmark Report for 1999,* Howard Rubin Associates, Pound Ridge. N.Y., 2000.
- Symons, Charles, *Software Sizing and Estimating Mk II Function Point Analysis*, John Wiley and Sons, 1992.

Letter to the Editor

Dear Editor:

It was so exciting to read through the articles in the November 2000 CrossTalk and see my company and division mentioned in *Acquistion Reform May Resemble Madness, but the Method is Real*. It feels great to be recognized for the work we have done at Raytheon, Command and Control Division here in Fullerton, Calif. Here are some corrections to the dates listed in the article for the Raytheon assessments:

- The Raytheon Command and Control Division, Fullerton, Calif., assessment was performed in October 1998. The press release was released in January 1999.
- The Raytheon Missile Systems, Software Engineering Center, Tucson, Ariz., assessment was performed in November 1998.

I was fortunate to be able to participate in both externally led assessments. Again, thank you for the article. It made my day!

Sally Cheung
Engineering Process Group
Integrated Systems Division,
formerly Command and Controls Division,
Raytheon

Applying Function Point Analysis to Requirements Completeness

Carol Dekkers
Quality Plus Technologies Inc.

Mauricio Aguiar
Caixa Economica Federal

Requirements issues abound in system development despite many models and methods intended to verify that requirements are complete. This article highlights how function point analysis (FPA), the software sizing technique, delivers value as a structured requirements review. While its historical usage has been confined almost exclusively to quantifying software size, FPA is gaining popularity as a useful, structured method for reviewing requirements. When used during software development to verify requirements completeness, FPA delivers more than mere numbers for software size—the FPA documentation reflects the full, known set of functional user requirements.

There is a wide diversity of traditional approaches for identifying and gathering software requirements, including joint application design sessions, requirements management techniques, prototyping, rapid application development, eXtreme Programming, and others. When done properly these techniques typically deliver a form of documented user requirements. After a series of user and peer reviews, these *formal* requirements are typically assumed to represent the complete set of user requirements.

However, the full set of user requirements is generally not complete until the project's end and continues to emerge as it progresses. As a result the project encounters rework, schedule slippage, and budget overruns, the extent of which depends on the degree of originally unknown requirements. With some Department of Defense projects this problem is further compounded due to requirements that are outcome- or performance-based, and functional requirements are developed as part of the design process. When these projects emerge, they challenge traditional requirements approaches. For example, how are software requirements documented when the performance requirement is to launch a projectile from a range of 200 miles with an impact of X? This article is not intended to solve these types of requirement issues.

This article addresses projects where user requirements are articulated (or should be) and outlines how function point analysis (FPA) can be an additional tool to identify missing requirements, gauge requirements completeness, and uncover potential defects. Our experience shows that FPA is often more effective than peer or user walkthroughs in identifying the full set of functional user requirements and uncovering potential defects. In fact, benefits gained by applying FPA to functional user requirements can be more valuable than the mere function point size of the software.

There are two audiences for this article:

1. Development teams that already use or are considering using FPA on their projects. The information provided here is intended to increase the cost-effectiveness of FPA, and leverage its use as a requirements completeness check.
2. Development teams that do not use FPA, but would like additional tools to increase requirements effectiveness.

The concepts outlined in this article can be applied to any project without the need to complete all steps in the method.

Requirements

Why is it that the *right*, i.e. correctly and accurately stated, set of software requirements is so elusive in our industry? Among various reasons, problems involve getting the requirements right, getting the right requirements (the complete set of functional user requirements), and often involve getting more than the specifications of requirements. One of the biggest problems software developers encounter is being able to judge whether requirements are sufficiently complete before beginning formal design and coding.

Before we discuss how to apply FPA to requirements completeness, it is worthwhile to identify the three major types of software requirements. Together, these form the overall project's user requirements. First are functional user requirements, which are the logical business or user functions the software must perform. All software from real-time missile guidance systems to business accounting software has functional requirements that must be performed. These include elementary processes that must be supported to input, process, manipulate, output, and interface data to, from, and within the software. FPA specifically addresses this type of user requirements.

Second are nonfunctional user requirements. These are the technology-independent user/business constraints that the software must meet. Nonfunctional requirements include quality and performance requirements such as portability, usability, security, dependability, reliability, and speed. Part of the FPA technique can assist with these types of requirements.

Lastly, technical requirements are the user requirements for a specific hardware/software configuration or a particular technical configuration that must be delivered. For example, the technical requirements may specify an Oracle database or a multi-tiered hardware solution. While these software specifications are as important as the other two types, FPA does not address this type of requirements.

The remainder of this article specifically pertains to functional and nonfunctional requirements.

Traditional Completeness Checks

While both the functional and nonfunctional requirements strive to be unambiguous, correct, and complete, it is easy to write down and check business rules for ambiguity and correctness with a user. The problem, however, is to ensure that the full set of functional user requirements has been identified. One or two frames of reference are needed. Such a frame of reference is typically provided using two existing techniques:

1. Theory-Based Model. A theory-based frame of reference may be used as structured analysis, information engineering, or data modeling. The analyst will decompose the problem and look for abstract structures like data flows, processes, and data stores. Requirements will be considered complete when the abstract structures make sense to the analyst, e.g., data stores have both incoming and outgoing data flows.

2. Personal Experience. The analyst may have worked with other business systems similar to the one being analyzed. In that case, he will possess a subjective frame of reference composed of all the business structures and rules he has previously encountered. The analyst will decompose the problem and look for known structures and rules conformant to his own model of reality. Requirements will be considered complete when the identified structures match the analyst's model of completeness, which is subjective, e.g., accounts receivable will have been either received or marked as delinquent.

Ordinarily, the analyst will work with a mixture of the first two frames of reference to increase quality and clarity of the documented set of known software requirements and to increase the relative percentage of the known to total requirements. Throughout the project he will integrate his personal experiences with the theoretical knowledge. Increasingly, however, this is insufficient to gain enough completeness coverage. It is in the analyst's interest to use as many frames of reference as possible.

Ideally, frames of reference should be orthogonal, i.e., they should not overlap. Each frame of reference should provide unique information not available in the other models.

Why Function Point Analysis?

Along with some of the nonfunctional requirements, FPA provides an additional frame of reference for checking the completeness of functional requirements. FPA is different from the first two frames of reference because it provides a unique, user-focused perspective. FPA examines the set of functional user requirements in terms of data and movement/manipulation (transactions) as understood and expressed by users; on this basis, it determines software's functional size. As such, FPA can be used in addition to the theory-based and personal experience models previously mentioned to ensure that functional user requirements are complete.

Function Point Basics

Function points (FPs) measure the size of a software project's logical user functionality as opposed to the physical implementation of those functions as measured by lines of code (LOC). FPA examines the functional user requirements to be supported or delivered by the software. It then assigns a weighted number of FPs to each logical user function as outlined in Function Point Counting Practices Manual [2] and calculates the software's FP size.

In simplest terms, FPs measure what the software must do from an external, user perspective irrespective of how the software is constructed. While analogies from other industries such as building construction and manufacturing attempt to describe how function point analysis works with software, none provides a perfect fit. In basic terms, FPs reflect the functional size of software, independent of the development language and physical implementation.

FPs can be likened to the functional area of a building by summing up its floor plan size. FPs quantify the functional user requirements (the floor plan) by summing up the size of its functional *components*. As with building construction, project management is not possible if only square foot size is known. System development cannot be managed purely on the basis of FP size.[1]

Using FPA to Gauge Completeness

For an introductory article on FPs, see the February 1999 issue of CrossTalk. When performing a FP count, all the known functional user requirements for the software are analyzed, weighted, and counted using the standard identification method. It is during analysis of functional user requirements that most errors and omissions in the requirements are uncovered, as described below. Following are the steps in the actual FP counting process:

1. Determine the project scope and purpose of the function point count. For example, FPs can be counted to quantify the size of a new development or enhancement/renovation project, or to size an existing base application.

In this step it is useful to document the specific name and date of the source document(s) used as a basis for the count (e.g., system ABC requirements document V1, March 22, 2000). This provides traceability of logical functions included within the functional requirements as a specific point in time, and is useful for gauging scope creep during the project. It can also contribute to the historical base for gauging future projects as outlined below.

By documenting—even in a few lines of text—the project scope and purpose of the FP count, project assumptions are clarified and requirements oversights identified. For example, if the purpose of the FP count is to size the amount of customization required for a commercial off-the-shelf package, the scope will include only the customized functions, not the entire package. This provides a delineation of what is included in the project.

2. Identify the application's logical boundary. This step identifies the functions that the software must perform, together with external users interfaces, departments, and other applications. The application boundary for FP counting is not the same as a physical one. Instead it is the logical boundary that envelops self-contained user functions that must exist to deliver the user requirements. This boundary separates the software from the user domain (users can be people, things, other software applications, departments, and other organizations). Software may span several physical platforms and include batch and on-line processes—all of which are included within the logical application boundary. For example, an accounts payable system would typically be considered one application in FPA, even though it may reside across multiple hardware platforms in its physical installation.

Because each *application* or software system has a separate application boundary (e.g., accounts payable would typically be one application, fixed assets may be another) a project context diagram consisting of several circles denoting various application boundaries is often drawn as a part of the functional sizing process. In cases where an enhancement project renovates an application that has little documentation, this step provides a context diagram that can be used later for communicating with the users about the software system. In a particular client situation, this visual depiction of various application boundaries and interfaced applications opens a discussion of client/server migration of certain applications because our diagrams showed which applications would be affected by the migration of a central application. Because these context diagrams are visual in nature and independent of technology, their review often leads to the discovery of interfaces that were previously discussed, but that are missing from the written requirements.

In addition, this step with subsequent steps, clearly demarcates logical boundaries between user applications. By clarifying which functions lie within which applications, there is less likelihood of a set of requirements being overlooked. For example, if a project team assumes that another application will maintain a set of common data, a review of the context diagram showing the interface to the other application may reveal potential oversights.

3. Count the Data Functions. This step considers internal and external data entities. It consists of:

- Identify, weigh, and count the internal logical files (ILFs). These are the persistent logical entities or data groups to be maintained through a standard function of the software.
- Identify, weigh, and count the external interface files (EIFs) that are persistent, logical entities referenced from other applications but not maintained. Typically these data are used in editing, validation, or reporting types of software processes.

When identifying and classifying the persistent logical entities as internal (maintained) and external (referenced-only), it is helpful to draw circles around the entities and their included subentities on a data model or entity-relationship diagram. If there is no data model or entity-relationship model, one is essentially created in this step by building on the context diagram created in the previous application boundary step.

Note that FPA does not count hard-coded data or any tables/files created only because of the physical or technical implementation. The data step records the number and types of logical data elements if they are known, and if they are not already identified in the requirements. This provides a checklist of data entities to gauge the consistency and completeness of transactional (manipulation of data) functions.

By reviewing the entities, whether on a data model or hand-drawn context diagram, and whether they are inside the application boundary (i.e., to be maintained by the software) or external (i.e., to be referenced only) often clarifies comments. Such comments might include: "Why is that entity external? I thought we needed to be able to update that entity." These would lead to a discussion that either confirms the original requirements or reveals an inconsistency in understanding and a change in the diagram. When the review is combined with the transactions outlined in the next step, the majority of (potential) requirements mismatches are identified.

4. Count the transactional functions. Use the following:

- External Inputs (EIs) that are the elementary processes whose primary intent is to maintain the data in one or more persistent logical entities or to control the behavior of the system. Note that these EIs are functional unit processes and not physical data flows or data structures.
- External Outputs that are the elementary processes whose primary intent is to deliver data out of the application boundary, and which include at least one of the following: mathematical calculation(s), derive new data elements, update an ILF, or direct the behavior of the system.
- External Queries that are the elementary processes whose primary intent is to deliver data out of the application boundary purely by retrieval from one or more of the ILFs or EIFs.

This step is where the majority of missed, incomplete, or inconsistent requirements are identified. This list provides some examples of the types of discoveries that can be made using FPA:

- If a persistent, logical entity has been identified as an ILF, i.e., maintained through a standard maintenance function of the application, but there are no associated EIs processes, there are one or more mismatched requirements:
  - Either the entity is actually a reference-only entity (in which case it would be an EIF), or
  - There is at least one missing requirement to maintain the entity, such as add entity, change entity, or delete entity.
- If there are data maintenance (or data administration) functions identified for data, but there is no persistent logical entity to house the data (ILF), the data model may be incomplete. This would indicate the need to revisit the data requirements of the application.
- If there is a data update function present for an entity identified as reference only (EIF), this would indicate that the entity is actually an ILF. The data requirements are inconsistent and need to be reviewed.
- If there are data entities that need to be referenced by one or more input, output, or query functions, and there is no such data source identified on the data model/entity-relationship diagram/context diagram, the data requirements are incomplete and need to be revisited.
- If there are output or query functions that specify data fields to be output or displayed that have no data source (i.e., no ILF or EIF), and the data is not hard-coded, there is a mismatch between the data model and the user functions. This indicates a need to revisit the data requirements.

Most maintained entities (ILFs) follow the Add, Update, Delete, Inquiry, Output (AUDIO) convention rule [3]; each persistent logical entity typically has a standard set of functions associated with it. Not all entities will follow this pattern, but AUDIO is a good checklist to use with the ILFs.

5. Evaluate the complexity of nonfunctional user constraints using a value adjustment factor. Through an evaluation of the 14 general systems characteristics (GSCs) of FPA (e.g., the GSCs include performance, end-user efficiency, transaction volumes, and others), a software complexity assessment can be made. The impact of user constraints in these areas is often not enunciated or even addressed until late in the software development life cycle, even though their influence can be major on the overall project.

Examining the user requirements with these nonfunctional, user business constraints in mind can provide the following types of valuable information:

- The nonfunctional requirement due to transaction rate peak loads may necessitate 24-hour, 7-day-a-week availability. This will have a critical impact on the resulting project.
- Special protection against data loss may be of critical importance to the users' business and must be specially designed into the system. This must be identified up front to avoid any unforeseen impact.

FPA provides an objective project size input for use in software estimation equations (together with other factors), or to normalize measurement ratios. The process checks whether the full set of functional user requirements has been identified and can uncover defective and missing requirements. Table 1 summarizes how to use FPA to uncover requirements defects. The far-right column of Table 1 illustrates where and what type of potential requirement problem there might be.

Benefits After the Requirements Phase

Having a documented set of functional user requirements (and the nonfunctional requirements that FPA addresses) such as that provided by the FPA process goes far beyond merely the requirements phase. Hill and Tinker Air Force Bases' Materiel Systems Groups (MSGs) found this to be the case. An example from Hill serves to illustrate this point: The MSG would attach a full listing of functional requirements (using the FPA-documented breakdown of FP components counted) to the software project estimate sent in to headquarters. Later, when questions arose about a particular set of functionality and whether it had been included, the group would refer to the FP listing to see if the particular functionality was listed. If it was not, it was clear that the functionality had not been included. A decision was then made about whether or not to include it and increase the estimate.

This simple set of documented functions minimized the finger pointing and blaming of "who said what and when," and reduced the discussion to whether or not the functions were included in the specifications submitted. Additionally, when scope changes emerged later in the project, as they inevitably do, both groups were in a position to adjust their FPA sizing and quickly assess the impact of scope change on the project.

While other requirements review and tracking techniques can also provide value, FPA is a simple method that delivers both a functional size of the software (useful for estimating) and can assist with the requirements processes.

Summary

Today's software analyst needs all the assistance he or she can find to help in the quest for complete (and known) user requirements. The framework provided by the structure of the FPA technique gives the analyst one extra frame of reference to gauge the completeness of the known user requirements. Requirements defects will still occur no matter how many frames of reference are used, however, the use of FPA to augment the traditional theory-based and personal experience frames of reference will increase the analyst's ability to ensure that software requirements are complete.

Is FPA worthy of your organization's consideration? The answer will vary depending on your organizational structure, goals, and measurement objectives. FPA is one tool that can assist with your requirements processes and also provide a quantitative value to size your software. For those of you who have been using FPA only to arrive at a software size, you can gain valuable benefits by applying FPA as a structured review, especially when your requirements are deemed *complete.*◆

Table 1. *Using FPA to Uncover Requirement Defects*

Data Function	ILF	EIF	Transaction Function	EI	EO	EQ	Requirement Problem Indicator	Potential Requirement Problem(s)
Employee Entity (maintained)	X	X					X	1. Same entity can't be maintained and externally referenced 2. No maintenance functions.
Monthly Sales	X		Add Sales Delete Sales	X X			X	Can errors be corrected/updated? Are there no reports that use or query data?
			Account Report			X	X	1. No data source in application identified containing account info. Where is data source?
% of FPA Component Breakdown	50%	0		10%		7%		Based on standard % profile questions include: 1. Why no external files – were they overlooked? 2. Why no queries in requirement when "standard" profile shows 10% of FP allocated to browse/query. 3. Are all transactional functions identified?
FPA Component Standard % Profile*	30%	10%		40%	10%	10%		

* The breakdown of standard percentages here is fictitious and intended to show a sample profile that could be developed using FP counts of a sample size of several similar applications.

References

1. Quality requirements can be found in the ISO/IEC 9126:2000 suite of standards that address many of the *ility* constraints such as portability, security, usability, reliability, etc. Contact ISO for further details.
2 The Function Point Counting Practices Manual (CPM) is maintained by the International Function Point Users Group (IFPUG) and is currently in Release 4.1 (1999).
3. Per personal discussions with John VanOrden, certified function point specialist, formerly of Gartner Group and a member of the Quality Plus Technologies Inc. consulting team. VanOrden uses the AUDIO checklist.

Note

1. When matters of software estimating are discussed, many more factors are involved beyond the functional size of software, including the type of software, technical requirements, number of users, geographic locations, etc.

About the Authors

Carol Dekkers is vice-chair of the Project Management Institute Metrics Special Interest Group. She is president of Quality Plus Technologies Inc., a management consulting firm specializing in helping DoD and private organizations succeed with function points, make wise investments in software measurement, and achieve bottom-line improvements through process improvement. Dekkers is a past president of the International Function Point Users Group and an International Organization for Standardization project editor on the Functional Size Measurement project. She was named one of the 21 New Faces of Quality for the 21st century by the American Society for Quality. She is a professional engineer, certified function point specialist, and a certified management consultant.

E-mail: Dekkers@qualityplustech.com

Mauricio Aguiar is a software manager with Caixa Economica Federal, a leading Brazilian government bank with more than 2,000 branches. He has 25 years' experience in software management, including the application of accelerated learning in information technology. Aguiar is president of the Brazilian Function Point Users Group and serves on the International Function Point Users Group (IFPUG) board of directors. A professional engineer and systems analyst with a master's degree in neuro-linguistic programming, he is a member of Project Management Institute, American Society for Quality, and the IFPUG.

E-mail: mauricioaguiar@yahoo.com

Measure Size, Complexity of Algorithms Using Function Points

Nancy Redgate
PRI Automation

Charles B. Tichenor
Defense Security Cooperation Agency

Software developers and software metrics analysts have long known that algorithms can be of important functionality in software applications. Examples of these algorithms' functions are: controlling a nuclear reactor, calculating complex pricing arrangements, optimizing production levels in manufacturing, and finding the shortest routes through transportation networks. These algorithms can require considerable effort to develop, and can contribute significantly to the size and complexity of the software. There have been a number of attempts to quantify the size and complexity of these algorithms using function point metrics; however, no such method is generally recognized as satisfactory. This leads to situations where functionality is only partly accounted for, or the current function point methodology is "patched up" in academically controversial ways. In contrast, our research shows that no "patches" are needed to account for many of these algorithms. We show how to identify them, disassemble them into functional components, and measure their size and complexity while remaining within the strict interpretation of current counting rules. Complete accounting of these algorithms leads to better software sizing accuracy, which produces better forecasts of costs, schedules, and measures of quality in terms of defects per function point delivered.

An algorithm is a set of equations that is executed in a logical sequence to produce an external output. In the fields of function point analysis and operations research, an algorithm can be seen as a critical tool to reduce effort needed to solve a complex series of calculations. We have written this paper focusing on intermediate function point analysis theory, and use linear programming to exemplify our points. Some readers may want to refer to the Definition of Terms at the end of this paper, or to [1, 2, and 3] for detailed references describing function point analysis, linear programming, and operations research, respectively.

An algorithm we counted recently was one used to compute pay. This required solving a set of equations to calculate such pay subcomponents as base pay rate, holiday pay, temporary duty (travel) expenses, and foreign exchange rates. All subtotals were added to yield the final pay amount. Using the method of the International Function Point Users Group (IFPUG), beginning function point counters would probably recognize the input screen needed to enter a traveler's pay/expense data and count six or fewer function points. They would probably recognize the resulting earnings/expense statement as an external output and count it as seven or fewer function points. However, they may overlook other, more substantial functionality inherent in this algorithm.

Conditions for Sizing an Algorithm Using Function Points*

- An algorithm must represent a step-by-step procedure based on mathematical calculations and perhaps logic statements. It contains a set of equations that are executed according to business rules. The solutions to the equations are stored until later combined to produce an external output (EO) the user can recognize.
- It must be complete, solvable, feasible, and should not contain redundant constraints. (If it contains redundant constraints, they are considered nonunique and, therefore, not countable.)
- It must have a logical storage area to hold solutions to intermediate equation calculations until they are finally combined into a meaningful EO. This logical storage area is counted as one or more internal logical files (ILFs). We count the number of data element types (DETs) as the number of unique algorithm variables that must be populated plus the number of unique instances of DET control information that must be used to operate the algorithm. We also count the number of record elements types (RETs) if the algorithm contains logical subgroups of data and/or control information.
- The ILFs must be maintained by externally inputting values of variables and of stored control information; therefore, there must be at least one external input (EI) for each algorithm. We count one DET for each ILF variable maintained and one for each instance of maintained control information. The file types referenced (FTRs) by the EIs include the algorithm's ILFs.
- Data in the algorithm's ILF must be used to perform one or more intermediate calculations. The result of this calculation sequence must be recognizable by the user. There must be at least one EO in the application, which contains the DETs resulting from this calculation sequence. We count the ILFs of the algorithm when determining the number of EO FTRs; moreoever, we include the number of unique DETs output from the algorithm in the number of EO DETs.
- As an option, the application may have an external inquiry (EQ) capability to permit the user to view the values of the algorithm's variables and control information. If so, we count the number of viewable variables and control information in the EQ DET count and count the number of ILFs touched by the EQ process as the number of FTRs.
- We also recognize that algorithms may exert a greater degree of functionality influence than software without algorithms. The function point general system characteristics (GSCs) must therefore also be considered during the analysis. Examples include:
  - GSC 5, On-Line Data Entry, may be influenced if the algorithm's variables are populated on-line.
  - Since there must be at least one algorithm ILF updated during real-time processing, GSC 8, On-Line Update, may need to be considered.
  - GSC 9, Complex Processing, is likely to be a characteristic affected by complex algorithms. The function point counter should examine the algorithm for functionality such as extensive logical and/or mathematical processing and adjust the function point count accordingly.
  - GSC 14, Facilitate Change, may need to be examined—especially the fifth item, "Business control data is kept in tables that are maintained by the user with on-line interactive processes and the changes take effect immediately."

* All definitions and conditions are as defined by International Function Point Users Group.

Example: Linear Programming

Formulating the Problem

To illustrate the algorithm function point counting process, a linear programming algorithm can serve as an example. This is because linear programming fits the criteria for an algorithm as previously described. Linear programming is a repeatable, step-by-step process based on mathematical calculations and logic statements depending on how the problem is solved.

Formulating and solving a linear programming problem requires externally inputting and storing the values of certain variables and instances of control information until they are needed, to perform intermediate calculations according to a logical sequence, and to externally output the solution. It is complete, solvable, feasible, and should not contain redundant constraints when well formulated (if a formulation contains redundant constraints, these are not counted).

We illustrate by performing a function point analysis of the Joseph Ecker's and Michael Kupferschmid's Brewery Problem [3]. Microbrewers Inc. makes four products called Light, Dark, Ale, and Premium. These products are made using the resources of water, malt, hops, and yeast. Microbrewers Inc. has a free supply of water, so it is the amount of other resources that restricts production capacity. Table 1 shows how these resources are used to produce each corresponding gallon of beer, the available amount of resources, and the revenue realized from selling each type. The problem is to calculate the product mix to brew to maximize revenue.

	Light	Dark	Ale	Premium	Available (lb.)
Malt	1	1	0	3	50
Hops	2	1	2	1	150
Yeast	1	1	1	4	80
Revenue($)	6	5	3	7	

Table 1. *Pounds of Resource Needed to Produce 1 Gallon of Beer*

We can formulate this problem using the Simplex linear programming algorithm methodology from the field of Operations Research. Following this methodology, we define each of the products as follows:

- X1 – Gallons of Light.
- X2 – Gallons of Dark.
- X3 – Gallons of Ale.
- X4 – Gallons of Premium.

We can also define the objective function as: Max: 6X1 + 5X2 + 3X3 + 7X4.

The resources are constrained by the available amount of each resource. Therefore, the objective function is subject to the following:

- X1 + X2 + 3X4 <= 50 (Malt)
- 2X1 + X2 + 2X3 + X4 <= 150 (Hops).
- X1 + X2 + X3 + 4X4 <= 80 (Yeast).

Also, each variable must be greater than or equal to 0, or, X1, X2, X3, X4 >= 0.

In its traditional Simplex form, then, this algorithm appears as follows:

Max: 6X1 + 5X2 + 3X3 + 7X4
Subject to:
X1 + X2 + 3 X4 <= 50 (Malt)
2X1 + X2 + 2 X3 + X4 <= 150 (Hops)
X1 + X2 + X3 + 4X4 <= 80 (Yeast)
X1, X2, X3, X4 >= 0

This formulation of the linear program could be considered the primal formulation. However, every linear program can be formed in both a primal and a dual formulation and each method produces identical results. Suppose we set the following as the resource variables:

- Y1 – Pounds of Malt.
- Y2 – Pounds of Hops.
- Y3 – Pounds of Yeast.

Then, the dual can be formulated as follows:

Min: 50Y1 + 150Y2 + 80Y3
Subject to:
Y1 + 2Y2 + Y3 >= 6 (Light)
Y1 + Y2 + Y3 >= 5 (Dark)
2Y2 + Y3 >= 3 (Ale)
3Y1 + Y2 + 4Y3 >= 7 (Premium)
Y1, Y2, Y3 >= 0

Counting the Unadjusted Function Points

Solving linear programs is an algorithmic procedure because a set of equations is executed in a logical sequence to produce an EO. Reaching the solution using the graphical method requires constructing several constraint lines, determining the feasible region, and then finding the corner point of the feasible region that optimizes the objective function. Solving using the Simplex method requires building *tableaus* according to a certain procedure until the optimal solution is found. Each data element type (DET) must be stored in a logical storage area until it is needed.

In this example, the logical data storage area consists of one ILF. This ILF is the set of equation variables and control information containing the objective function and constraints. This meets the test of an ILF because it will be demonstrated that it is a logical, user-identifiable group of data or control information; and the group of data is maintained through an elementary process within the counted application boundary. ILFs have record element types (RETs) and DETs as components. Here is our approach for determining their number in this algorithm.

RETs

In a linear program, there are several logically distinct subgroups of data. The first is the objective function. It contains the control information telling us to either maximize or minimize. It also contains the coefficients for the decision variables that, in this example's primal formulation, indicate the revenue corresponding to each product. The other subgroups are represented by the constraint equations. Graphically, each constraint equation represents a unique set of data points in the plane (or space) that are feasible contributions to the solution of the linear program.

Either the primal or dual formulation of a linear program is mathematically sound. However, one may have fewer constraint equations. Extending the notion of the *elementary process* we always choose the smallest unit of activity meaningful to the user. In principle this means counting the formula-

tion containing the fewest constraint equations. In this example, it is the primal:

- We count five RETs in this algorithm.
- We count one RET for the optimization function.
- We count the minimum number of variables and number of constraints. This guarantees that the same number of RETs is counted regardless of the primal or dual formulation of the problem. This is three RETs since the primal formulation is the smaller unit of activity according to our reasoning.
- We count one RET for the nonnegativity constraint.

DETs

- We count 24 DETs in the algorithm.
- We count one DET for the maximum (or minimum) function, which is an instance of control information telling us how to optimize the objective function.
- We count one DET for each nonnegative variable or constraint. This is analogous to the number of variables in Table 1. In this example we count 18 such DETs.
- We count one DET for the number of variables and number of constraints in the primal or dual formulation used to count RETs, and we add one for the zero. This accounts for the nonnegativity condition. In this case there are five such DETs (X1, X2, X3, X4, and zero) for the nonnegativity constraint.

The unadjusted function point count of this ILF of five RETs and 24 DETs is 10, and is therefore an *average* ILF.

To maintain the data in the ILF, we must use an EI. In this example, there is one FTR and there are 24 DETs (one for each data element plus one for minimum/maximum function). If this were a simple EI, there would probably be one further DET representing invoking the *enter* key to initiate the EI process and perhaps another DET if there were an associated error/confirmation message. This would be an average EI and would contribute four unadjusted function points.

The number of EOs will vary depending on the user requirements. Suppose the user wanted an output screen showing the optimal value of the objective function for this linear program, the optimal assignment for each of the variables, and the shadow price of each variable:

- We count one DET for the optimal value of the objective function, or one.
- We count one DET for each optimal variable assignment, or four.
- We count one DET for each shadow price or three.

To show all three of these aspects we count eight DETs. Since there is one FTR this would be a low EO of four unadjusted function points.

In this example, we count the total unadjusted function points as 18:

10 (ILF) + 4 (EI) + 4 (EO) = 18

Solving linear programs is an algorithmic procedure because a set of equations is executed in a logical sequence to produce an external output.

Counting Large Algorithms

Some algorithms contain a few variables, like the preceding example; however, algorithms can contain many hundreds of variables. If the function point counter believes that there is more functionality inherent when counting these types of algorithms, then the counter may want to consider the super file rule (SFR).

A *super file* is defined as an ILF or ELF that contains more than 100 DETs if it contains multiple, countable RETs. If this is the case, each RET is considered a unique ILF (or EIF) and is counted as such. Although the SFR is not recognized by IFPUG, it is statistically significant. Some organizations informally adopt the SFR as part of their own local counting practice resolutions, and footnote their counting documentation accordingly.

Conclusion

An algorithm is a set of equations that are executed in a logical sequence to produce an external output. Algorithms appear in a variety of software applications. They can be used to help production planning in a brewery, perform many sets of calculations in sales applications, control nuclear reactors, schedule training, and determine the shortest route for telephone calls through a city telephone line network. The widely accepted IFPUG function point methodology can be used to count many algorithms. The paradigm described in this paper requires breaking down an algorithm into its functional components. These include its ILFs, EIs, and EOs. It also includes examining the corresponding GSCs. This paradigm is repeatable and reliable.

Recognizing and counting these kinds of algorithms is important. Their function point count helps quantify the work effort required to develop them that otherwise would have been overlooked. It also better portrays the size and complexity of the software. Finally, it helps quantify better cost and schedule forecasts, and can improve software quality measurement for overall software development.◆

References

1. International Function Point Users Group (IFPUG), *Function Point Counting Practices Manual Release 4.1.,* 1999
2. *Function Point Counting Practices Manual 3.4,* 1991.
3. Ecker, Joseph G. and Kupferschmid, Michael, *Introduction to Operations Research*, Malabar, Fla., Krieger Publishing Company, 1988.
4. Ibid., pp. 16-17.
5. Sedgewick, Robert, *Algorithms,* Addison-Wesley Publishing Co., Reading, Mass., 1983.

Visit www.ifpug.org for more information on International Function Points User Group.

Additional Reading

- Garmus, David and Herron, David, *Measuring the Software Process: A Practical Guide to Functional Measurements,* Upper Saddle River, N.J., Prentice Hall PTR, 1996.
- Jones, Capers, *Applied Software Measurement: Assuring Productivity and Quality,* New York, McGraw-Hill, Inc., 1991.
- Monks, Joseph G. *Operations Management,* New York, McGraw-Hill Publishing Company, 1985.
- Shapiro, Roy D., *Optimization Models for Planning and Allocation: Text and Cases in Mathematical Programming,* New York: John Wiley & Sons, 1984.

Definition of Terms

Note: Some of these terms have two definitions. We have provided explanations in layman's terms. The italicized definitions are the precise ones from the IFPUG Counting Practices Manual.

Algorithm. An algorithm is the set of rules that must be completely expressed in order to solve a significant computational problem [4].

Application. This is a software package, such as a word processing, spreadsheet, or checkbook package.

Application User (simply referred to as "user"). A user is someone who needs a software application to perform his or her duties. For example, a user set might include data entry clerks, managers who need certain reports, customers who receive bills, system administrators who need to query the software's databases, et al. A user set does not normally refer to those whose role is software production such as programmers, database designers, or release managers; their role is to develop the software, not to use it after its market implementation.

Data Element Type (DET). Usually a DET is a field of data. It can also be an element of control information, such as the Enter key when it is needed to initiate the process of data input into an internal data file. In general, the more DETs in a function type (such as an external input), the higher its function point size. *"A unique, user-recognizable, nonrecursive field. The number of DETs is used to determine the complexity of each function type and the function type's contribution to the unadjusted function point count."*

Dual. This is a certain perspective of defining the resources available to reach the stated objective in linear programming. It is essentially a reflection of the primal perspective.

External Input (EI). EI is the process of adding, changing, and/or deleting data from an internal database. An example would be entering check numbers and amounts into a checkbook software package. An EI has three, four, or six unadjusted function points depending on whether it is of low, average, or high size/complexity. The textbook definition includes *"... processes data or control information that comes from outside the application's boundary. The external input itself is an elementary process. The processed data maintains one or more [internal logical files] ILFs. The processed control information may or may not maintain an ILF."*

External Inquiry (EQ). The process that allows the user to simply read or retrieve existing data from a database using certain criteria, much like an automated card catalog system in a public library. An EQ has three, four, or six unadjusted function points depending on whether it is of low, average, or high size and complexity. The textbook definition includes *"... an elementary process made up of an input-output combination that results in data retrieval. The output side contains no derived data. No ILF is maintained during processing."*

External Interface File (EIF). A database maintained in another application, but accessed by the application being counted on a read-only basis. An EIF has five, seven, or 10 unadjusted function points depending on whether it is of low, average, or high size/complexity. The textbook definition includes *"... a use-identifiable group of logically related data or control referenced by the application, but maintained within the boundary of another application. This means an EIF counted for an application must be an ILF in another application."*

External Output (EO). The process that yields a completed report, output file, or any other type of message set, which is sent to users. The report often contains data in fields that require calculations to derive. Examples could include credit card bills, completed spreadsheet reports, or state tax refunds. An EO has four, five, or seven unadjusted function points depending on whether it is of low, average, or high size and complexity. The textbook definition includes *"... is an elementary process that generates data or control information sent outside the application's boundary."*

Definition of Terms for this article is continued on page 30.

About the Authors

Nancy Redgate has a bachelor's degree in industrial engineering/operations research from the University of Massachusetts at Amherst. She received master's degrees in operations research, statistics, and business administration from Rensselaer Polytechnic Institute (RPI). She was the primary author of this paper, submitted as the requirement for an independent study in operations research at RPI.

PRI Automation
805 Middlesex Turnpike
Billerica, Mass. 01821
Voice: 978-670-4270
E-mail: nancy.redgate@prodigy.net

Charles B. Tichenor has a bachelor's degree in business administration from Ohio State University, a master's degree in business administration from Virginia Polytechnic and State University, and a doctorate degree in business from Berne University. He serves as an information technology operations research analyst for the Department of Defense, Defense Security Cooperation Agency. Tichenor holds part-time positions as a senior consultant for Development Support Center in Elm Grove, Wis., and as an adjunct faculty member at Strayer University's Anne Arundel, Md. campus. He served as technical advisor for this paper.

Defense Security Cooperation Agency
1111 Jefferson Davis Hwy., Suite 303
Arlington,Va. 22202-4306
Voice: 703-601-3746
Fax: 703-602-7836
E-mail: tichenor@erols.com
Internet: tichenor@erols.com

The Nine-Step Metrics Program

Timothy K. Perkins
Software Technology Support Center

Metrics are essential in evaluating program or project performance. However, several organizations remain confused regarding what measurements to collect, and how to use the measurements after they are collected. Before addressing these questions, some terms need to be defined. For the purpose of this article, a metric is defined as a combination of two or more measurements. Measurements are the raw data gathered from comparing an entity to a standard. After reading the article, feel free to use your own definition of the above terms.

While evaluating and commenting on the Measurement and Analysis Process Area of the Software Engineering Institute's (SEI) Capability Maturity Model Integration (CMMI^SM) version 0.2, members of the Software Technology Support Center at Hill Air Force Base, Utah, developed a nine-step measurement process with the steps logically grouped by activity type. The three activity groups are measurement planning, measurement implementation, and measurement program evaluation. Following is a presentation of these steps.

Activity Group 1

Measurement Planning

There are four measurement planning activities, the results of which are documented in the measurement plan. The following are the four activities:

1. Define Information Needs.

All measurements should adhere to the following criteria:

- Criterion 1 – Measurements should induce the parts to do what is good for the system as a whole.
- Criterion 2 – Measurements should direct managers to the point that needs their attention [1].

These criteria support the goal–question–metric (GQM) paradigm developed by Victor Basili. The key concepts of this paradigm are:

- Processes (software development, program management, acquisition management, etc.) have associated goals.
- Each goal leads to one or more questions regarding the accomplishment of the goal.
- Each question leads to one or more metrics that will answer the question.
- Each metric requires two or more measurements needed to create the metric.
- Measurements should be selected that provide the data needed to create the metrics necessary to answer the questions that determine goal accomplishment.

Eliyahu Goldratt supports the GQM paradigm in his statement "Measurements are a direct result of the chosen goal. There is no way that we can select a set of measurements before the goal is defined [2]."

Another point to remember, according to Joseph Juran, is that different organizational levels require different metrics. At the worker level, measurements are usually taken in terms of deeds performed or in things produced, e.g., how many, how much, or physical things (time, mass, space). Top level managers usually speak in terms of dollars—the impact on the bottom line. Those in the middle must be capable of communicating using both frames of reference. For example, the company financial statement is in the language of dollars. The sales forecasts and the results are in dollars and units. The production schedules, order points, and material requisitions are all in units [3].

2. Define Metrics and Analysis Methods.

This step is a continuation of the GQM paradigm described above, with the additional task of defining the analysis methods that will be used to create information from the data collected. The topic of proper data analysis is not trivial, and is well beyond what can be covered in a short article. The handbook [4] is an excellent starting point regarding measurement in general and includes several chapters that discuss analyzing data. Best of all, it can be downloaded free from the SEI Web site [www.sei.cmu.edu].

3. Define the Selected Measures.

This is the final step of the GQM paradigm. The selected measures are chosen not only to provide the information needed to answer the questions, but also to allow analysis using the methods determined in Step 2 above. Measurements used to characterize process performance should:

- Relate closely to the issue under study.
- Have high information content.
- Pass a reality test.
- Permit easy and economical collection of data.
- Permit consistently collected, well-defined data.
- Show measurable variation.
- Have diagnostic value as a set [4].

Let's look at each of the above points in some depth.

Relate closely to the issue under study. As mentioned in the paragraph discussing GQM, measurements must enable us to answer the questions related to individual goals.

Have high information content. A single measurement that provides a significant amount of information is more valuable than a set of three, four, or more measurements required providing the same information.

Pass a reality test. Does the proposed measurement really provide information necessary to answer a question regarding a goal? Or is it just a *feel-good* measurement that has been collected traditionally, but offers no real value?

Permit easy and economical collection of data. This is one goal of a measurement program. Data that are readily available and answer the questions regarding a goal are more desirable than similar data that are difficult or expensive to gather. Do not hesitate to perform a cost benefit analysis regarding data that appear to be difficult or expensive to collect.

Permit consistently collected, well-defined data. Once again, repeatability of data is the reason for strict identification of collection points.

Show measurable variation. Data that does not exhibit variation is useless in determining how to improve a process. It is the range of the variation that determines whether or not a process is under statistical control and indicates whether or not process changes are achieving desired results.

As a set, have diagnostic value. As stated, measurements combine to form metrics that are used to answer questions regarding goals. The set of measurements selected must provide the information needed to determine goal accomplishment, otherwise the measurement set is insufficient. According to Florac et al., "They should be able to help you identify not only that something unusual has happened, but what might be causing it [4]."

In the process of selecting measurements, do not forget to spend some time determining how collected data will be analyzed. Some analysis techniques require a certain volume of data collected at regular frequencies with a minimum level of accuracy in order to provide meaningful results. Make sure you understand how the data will be analyzed and plan accordingly.

4. Define the Collection Process of Measurement Data.

The points in the process where measurements are to be collected should be identified. Are measurements to be taken before or after a certain procedure has been performed, prior to or subsequent to certain integration efforts, etc.? Additionally, the manner whereby data is to be collected and the individual responsible for collecting the data should be specified by job title. If at all possible, the data generation should be a normal part of or step in the process.

Measurements must be clearly defined. This definition should explicitly state what is included in and excluded from the measurement. This allows those who use the data to thoroughly understand what the data represent, to permit the repetition of data collection, and to compare data samples. A good example of the necessity of clearly defining measurements is to ask a group of individuals to determine the number of lines of code in a short program listing. Depending on the language, arguments can be made regarding control code, comment lines, multiple executable statements on a single line, etc.

The frequency of data collection also needs to be specified. Measurements should be taken frequently enough to identify problems, and allow their correction prior to generating substantial scrap, creating substantial rework, or missing critical milestones. For example, if an organization cannot afford to lose the month-long effort of five individuals working on a project prior to identifying a problem in product production, measurements frequency must be substantially more than monthly. In determining the collection frequency, do not forget to include the time required to process the data into measurements and metrics, and to get the metrics into decision-makers' hands.

The Nine Steps in Action – An Example

The following example of how the steps are used in a measurement program is based on the idea that an organization has been tasked to deliver a new software release within 120 days. See [5] for an example of how the desired metric is calculated.

Step 1: Define information needs. Suppose one of the goals of the organization was to deliver the product on time. Questions regarding this goal are:

- Does the schedule estimate allow sufficient time to produce the product, or is the schedule artificially constrained?
- How much time is allocated for product development?
- Is there sufficient staff to provide the estimated needed hours within the required schedule?
- How much work has been accomplished on the critical path?
- How much work should have been accomplished on the critical path?
- How much time is remaining?

Step 2: Define metrics and analysis methods to address information needs. For the sake of this example, let's look at the third bullet from Step 1: Is there sufficient staff to provide the estimated needed hours within the required schedule? The metric used may be staff hours available per day. The analysis method chosen could be the use of X-Bar and R charts to determine if the number of hours delivered per day is within statistical control. The term "staff hours available per day" should be explicitly defined so that everyone on the project understands what is meant by an available staff hour.

Step 3: Define the selected measures. The measure to be collected would be the number of productive hours per person assigned to the project per day. For example, time spent in a team meeting discussing the project may be included while time spent answering e-mail on an unrelated project may not be included.

Step 4: Define the collection process of the measurement data. The collection process is to record the time reported against the project per person per day.

Activity Group 2

Measurement Implementation

The next activity group is measurement implementation. Continue with the following procedures:

5. Collect the Measurement Data.

This activity is simply the execution of the measurement data collection process methods.

6. Analyze the Measurement Data to Derive Metrics.

Metrics are derived from the analysis performed on the measurement data. The quality of the metric is tied to the rigor of the analysis process and the quality of the data collected.

7. Manage the Measurement Data and Metrics.

The measurement data and metrics must be managed properly according to the requirements defined in the measurement plan.

8. Report the Metrics.

Once the metrics are derived from the analysis of the measurements, they are made available to all those either affected by the metrics or by the decisions made because of the metrics.

Continuing the Example

Step 5: Collect the measurement data. The hours per day accomplished on the project per person is collected from the time cards the workers complete daily.

Step 6: Analyze the measurement data to derive metrics. Daily measurements are used to calculate additional points on X-Bar

and R charts, which are then analyzed to determine if the process is in statistical control and to determine if the average number of hours delivered are equal to or greater than the average number expected. Remember that the process can be within statistical control without meeting the average number of hours needed.

Step 7: Manage the measurement data and metrics. Archive the data in a manner that it can be readily retrieved, if needed.

Step 8: Report the metrics. Report whether or not the process is in statistical control and whether or not the necessary number of hours is being delivered.

Activity Group 3

Measurement Program Evaluation

9. Review the usability of the selected metrics.

Initially, the selection of metrics, analysis methods, and specific measurement data may be a *best guess*. Whether or not they meet specified information needs must be determined by experience. Over time, through a review of the usefulness of the metrics, the selection can be refined to a high correlation between the metrics selected and the information needs. This will be an iterative process.

The old adage "keep it simple" is a good rule to follow when establishing a metrics program. Remember to focus the metrics on the organization's goals. Because an organization will have a limited number of goals, there should be a limited number of necessary metrics. In other words, do not go overboard in collecting all potential measures. Collect only those necessary to determine goal achievement.

In a study performed in the early 1990s, Rifkin and Cox sampled organizations that had excellent software measurement practices. Their finding was that none of the organizations sampled had more than a dozen metrics [5]. Start small, collect and evaluate the data, and make changes as necessary. It is better to implement the 70 percent solution and evolve to the 100 percent solution rather than allow the analysis paralysis of trying to hit 100 percent on the first try to keep the organization from implementing anything.

Continuing the Example

Step 9: Review the usability of the selected metrics. After analyzing and reporting on staff hours per day, you may realize those hours alone are not a sufficient metric. Maybe productivity should be included, e.g., how much work on the project is being accomplished for each hour delivered. This would involve measuring task accomplishment per hour delivered.

A Caution in Selecting Metrics

Goldratt offers the following caution regarding measurements, "Tell me how you measure me, and I will tell you how I will behave. If you measure me in an illogical way… do not complain about illogical behavior [6]." This implies that the measurements you take may cause individuals within an organization, or an organization as a whole, to behave in a given manner.

To illustrate, Goldratt uses the example of a steel mill that was losing money where the primary performance measurement was tons per hour. At the end of the month, when the monthly measurement of tons per hour was coming due, the organization concentrated on producing *tons* of steel without regard to customer orders. In the rolling operation, thick steel plate took less time to produce than thin steel plate. Can you guess the thickness of plate produced? Customer orders went unfulfilled, and customers complained, but to no avail. Only when the measurement was changed from tons per hour to orders satisfied did the steel mill begin to use black ink rather than red ink [7]. Make sure the measurements you select cause the organization to behave in the desired manner.

Goldratt further warns, "Change my measurements to new ones that I don't fully comprehend and nobody knows how I will behave, not even me [8]." Make sure the organization's members understand the reason for the measurements and how they will be used before attempting to institute the collection of a set of new measurements.◆

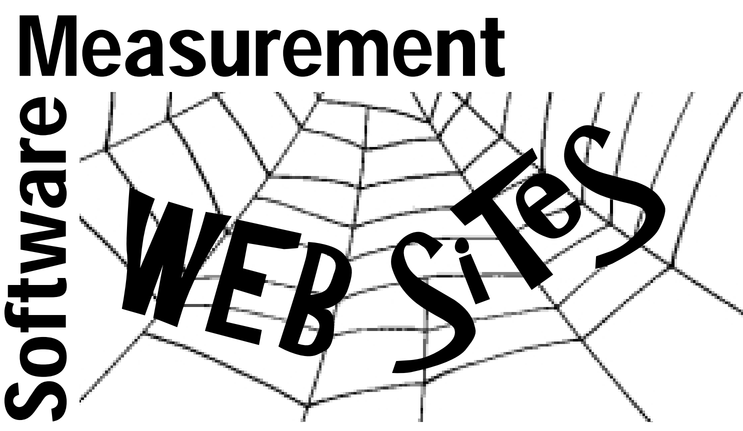
References

1. Goldratt, Eliyahu M., *Critical Chain*, The North River Press Publishing Corp., Great Barrington Mass, 1997, pp. 81-82.
2. Goldratt, Eliyahu M., *The Haystack Syndrome*, North River Press, Croton-on-Hudson, N.Y., 1990, p. 14.
3. Juran, Joseph M., *Managerial Breakthorugh*, 30th Anniversary Edition, McGraw-Hill, New York N.Y., 1995, pp. 240-241.
4. Florac, William A., Park, Robert E., Carlton, Anita D., *Practical Software Measurement: Measuring for Process Management and Improvement* (CMU/SEI-97-HB-003), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, 1997, p. 19
5. Rifkin, S. and Cox C., *Measurement in Practice* (CMU/SEI-91-TR-016), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, 1991.
6. Goldratt, *The Haystack Syndrome*, p. 28.
7. Goldratt, *Critical Chain*, pp. 104-112.
8. Goldratt, *The Haystack Syndrome*, p. 88.

About the Author

Timothy K. Perkins has been involved in software process improvement for the past 11 years, since he led the effort to initiate the software process improvement effort at the (then) five Air Force Air Logistics Centers. As the Software Engineering Process Group leader at the Software Engineering Division at Hill Air Force Base, Utah, he led the division in reaching CMM Level 3. The division has gone on to achieve CMM Level 5. Since retiring from the Air Force, he has been employed by Science Applications International Corp. as a process improvement consultant currently under contract with the Software Technology Support Center, which provides consulting services to Air Force and other DoD and government agencies. Perkins holds a bachelor's degree in electrical engineering from Brigham Young University and a master's degree in business administration from the University of Phoenix.

OO-ALC/TISE
7278 4th Street
Hill AFB, Utah 84056
Voice: 801-775-5736
Fax: 801-777-8069
E-mail: tim.perkins@hill.af.mil



International Council on Systems Engineering
www.incose.org
The International Council on Systems Engineering (INCOSE) is a not-for-profit membership organization founded in 1990 to develop, nurture, and enhance the system engineering approach to multidisciplinary system product development. The organization works with industry, academia, and government to disseminate systems engineering knowledge, promote education and research, and establish standards. The Web site lists conferences, workshops, seminars and courses, and features bulletins, technical journals, and electronic bulletin boards on systems engineering. INCOSE serves more than 3,200 multinational systems engineering professionals.

International Function Point Users' Group
www.ifpug.org
The International Function Point Users' Group (IFPUG) is a nonprofit organization committed to increasing the effectiveness of its members' IT environments through the application of function point analysis (FPA) and other software measurement techniques. IFPUG endorses FPA as its standard methodology for software sizing and maintains the Function Point Counting Practices Manual, the recognized industry standard for FPA. It also provides a forum for networking and information exchange, and provides an annual conference, educational seminars and workshops, professional certification, industry publications and more. IFPUG serves more than 1,200 members in more than 30 countries.

Practical Software and Systems Measurement Support Center
www.psmsc.com
Practical Software and System Measurement is a U.S. Army site-sponsored by the Department of Defense (DoD). The goal of the project is to provide project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs. PSM is based on actual measurement experience with DoD, government, and industry programs. Measurement professionals from a wide variety of organizations participate in the project, which includes systems and product engineering as well as software measurement. The Web site has the most current version of the PSM Guidebook, and is being updated to include an online search, a user's forum information exchange and automatic notification of product enhancements for registered users.

Software Metrics Sites
http://user.cs.tu-berlin.de/~fetcke/metrics-sites.html
The Software Metrics Sites are a guide to Internet resources on software measurement, process improvement, and related areas. Topics featured include electronic papers, bibliographies, and conferences on software measurement, object-oriented metrics, function point analysis, and software process improvement. The Software Metrics Sites list research institutes and people who are active in the area of software measurement. Several mailing lists that are used for discussions and ideas exchange can be found as well as software measurement tools that are available for download.

Association for Computing Machinery
www.acm.org
The Association for Computing Machinery (ACM), is an international scientific and educational organization dedicated to advancing IT arts, sciences, and applications. With a worldwide membership of 80,000, ACM functions as a locus for computing professionals and students working in the various IT fields. The site features news and publications, conference listings, and a library.

Computer Measurement Group Inc.
www.cmg.org
The Computer Measurement Group (CMG) is a nonprofit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with two areas: performance evaluation of existing systems to maximize performance (e.g. response time, throughput, etc.); capacity management where planned enhancements to existing systems or the design of new systems are evaluated to find the necessary resources required to provide adequate performance at a reasonable cost.

The Institute of Measurement and Control
www.instmc.org.uk
The Institute of Measurement and Control brings together thinkers and practitioners from many disciplines that have a common interest in measurement and control. It organizes meetings, seminars, exhibitions, and national and international conferences on a large number of topics. It has a very strong level of local section activity, providing opportunities for interchange of experience and for introducing advances in theory and application. It provides qualifications in a rapidly growing profession and is one of the few chartered engineering institutions that qualifies incorporated engineers and engineering technicians as well as chartered engineers.

Society for Software Quality
www.ssq.org
The Society for Software Quality (SSQ) promotes increased knowledge and interest in quality software development and maintenance technology. Its charter is to advance the arts, sciences, and technologies of quality software and to nurture and promote professionalism in those who engage in these pursuits. The SSQ is a federally recognized public benefit corporation organized and operated exclusively for educational purposes. It is dedicated to improving software quality and to providing communication between academia, industry, and software professionals.

Software Productivity Consortium
www.software.org
The Software Productivity Consortium is a unique, nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. The site features an interactive section to discuss new trends.

CMM Level 4 Quantitative Analysis and Defect Prevention

Al Florence
MITRE Corp.

The Software Engineering Institute's Software Capability Maturity Model® (SW-CMM) Level 4 quantitative analysis leads to SW-CMM Level 5 activities. Level 4 Software Quality Management (SQM) key process area analysis, which focuses on product quality, feeds the activities required to comply with defect prevention (DP) at Level 5 [1]. Quantitative Process Management (QPM) at Level 4 focuses on the process that leads to technology change management and process change management at Level 5. At Level 3, metrics are collected, analyzed, and used to status development and to make corrections to development efforts, as necessary. At Level 4, measurements are quantitatively analyzed to control process performance of the project and to develop a quantitative understanding of the quality of products to achieve specific quality goals. This paper presents the application of statistical process control (SPC) to accomplish the SQM and QPM and apply these results to DP. Real project results are used to demonstrate the use of SPC as applied to software development. An overview of control charts is presented along with Level 4 quality goals and plans to meet these goals.

An organization performing Level 4 quantitative analysis recognizes that it leads to Level 5 activities. This article presents this progressive relationship in project examples where statistical process control (SPC) is used to analyze measurements. Results of this analysis are used to gain a quantitative understanding of process capability, manage progress toward achieving quality goals, and for defect prevention.

The project used here is a large information system with a database of more than 30 million records developed on networked, client/server, workstations, and a multiprocessor parallel server utilizing C, UNIX, and ORACLE. The project was developed during five years with a staff of more than 100 with 300,000 lines of code and many commercial off-the-shelf packages. The project had achieved Level 3 in the SW-CMM, and the organization was pursuing Level 4. All Level 4 and Level 5 processes were installed and conducted on the project during a period of time.

This author was the software manager of the project at the time Level 3 was achieved. The author was also the SEPG lead during Level 4 activities and developed and installed Level 4 and Level 5 processes on the project.

The main quantitative tool used was SPC, utilizing control charts along with other methods. The project analyzed life-cycle data collected during development for requirements, design, coding, integration, and during testing. Defects were collected during these life-cycle phases and were quantitatively analyzed using statistical methods. The intent was to use this analysis to support the project in developing and delivering high quality products, and at the same time use the information to make improvements, as required, to the development process.

Rigorous statistics have been used in manufacturing but have had limited use in software development. The SEI's Capability Maturity Model Integrated℠ (CMMI) calls for rigorous statistics at Level 4 and emphasizes SPC. This paper shows that control charts and other statistical methods can easily and effectively be applied in a software setting.

Control Chart Analysis

Figure 1 demonstrates how control charts are used for this analysis [2]. According to the normal distribution, 99 percent of all normal random values lie within +/- 3 standard deviations from the norm, 3-sigma [3]. If a process is mature and under SPC, 99 percent of all events should lie within the upper and lower control limits. If an event falls out of the control limits the process is said to be out of SPC; the reason for this anomaly needs to be investigated for cause, and the process brought back under control.

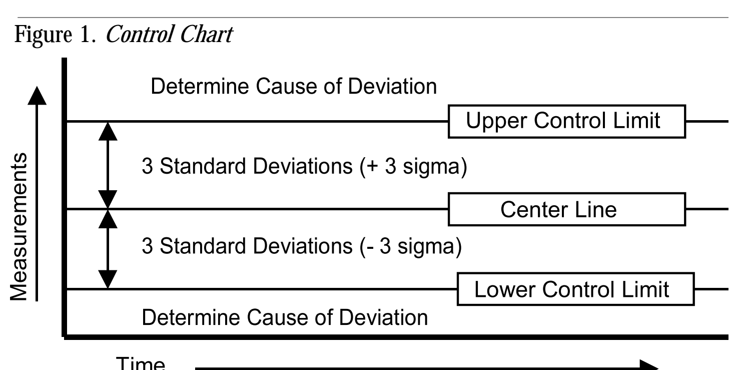
Control charts are used because they separate signal from noise. Thus when anomalies occur they can be recognized. They identify undesirable trends and point out fixable problems and potential process improvements. Control charts show the capability of the process, so achievable goals can be set. They provide evidence of process stability, which justifies predicting process performance [2].

Control charts use variables and attributes data. Variables data are usually measurements of continuous phenomena. Examples of variables data in software settings are elapsed time, effort expanded, and memory/CPU utilization. Attributes data are usually measurements of discrete phenomena such as number of defects, number of source statements, and number of people. Most measurements in software used for SPC are attributes data. It is important to use the correct data on a particular type of control chart [2].

When attributes data are used for direct comparisons, they must be based on consistent *areas of opportunity* if the comparisons are to be meaningful. In general, when the areas of opportunity for observing a specific event are not equal or nearly so, the chances for observing the event will differ across the observations. When this happens, dividing each count by its area of opportunity normalizes the number of occurrences [4].

Charts for averages (X-bar charts) and range (R charts) are

Figure 1. *Control Chart*



Capability Maturity Model Integrated (CMMI) is a service mark of the Software Engineering Institute and Carnegie Mellon University.

used to portray process behavior when the option exists to collect multiple measurements within a short period of time under basically the same conditions. When the data are collected as such, measurements of product or process characteristics are grouped into self-consistent sets (subgroups) that can reasonably be expected to contain only common cause variation. The results of the subgroups are used to calculate process control limits, which in turn are used to examine stability and control the process [4].

The following is a list of control charts that should be used for variable data and for attributes data:

Attributes Data:	Variable Data:
• u charts	• X-bar charts
• Z charts	• R charts
• XmR charts	• XmR charts

Level 4 Leads to Level 5

Figure 2 shows how data collection, analysis, and management from Level 4 activities leads to Level 5 activities of defect prevention, technology change management (TCM), and process change management (PCM) [5].

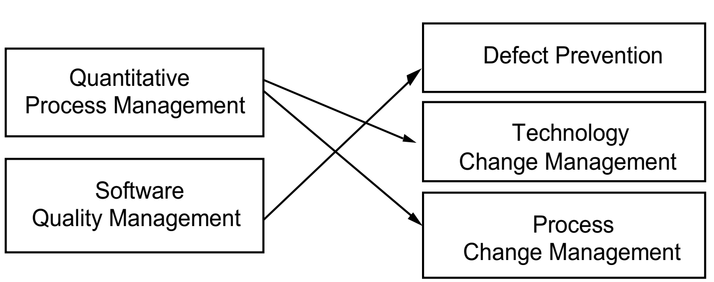


Figure 2. *Level 4 and Level 5 Paths of Influence*

Quantitative process management (QPM),which focuses on the process, leads to making process and technology improvements. Meanwhile software quality management, which focuses on quality, leads to preventing defects.

Level 4 Goals and Plans

The Capability Maturity Model V1.1 requires that Level 4 quality goals, and plans to meet those goals, be based on the processes implemented, that is, on the processes' proven ability to perform. Goals and plans must also reflect contract requirements. As the project's process capabilities and/or contract requirements change, the goals and plans may need to be adjusted.

The project this paper is based on had the following key requirements:

- Timing: subject search response in less than 2.8 seconds 98 percent of time.
- Availability: 99.86 percent seven days, 24 hours.

These are driving requirements that constrain hardware and software architecture and design. To satisfy these requirements, the system needs to be highly reliable and have sufficiently fast hardware.

The quality goals are:

- Deliver a near defect-free system.
- Meet all critical computer performance goals.

The plans to meet these goals are:

- Defect detection and removal during:
  - Requirements peer reviews.
  - Design peer reviews.
  - Code peer reviews.
  - Unit tests.
  - Thread tests.
  - Integration and test.
  - Formal tests.
- Critical computer resource monitoring:
  - General purpose million instructions per second (MIPS).
  - Disc storage read inputs/outputs per second (IOPS).
  - Write IOPS per volume.
  - Operational availability.
  - Peak response time.
  - Server loading.

The following quantitative analyses are real project examples applying SPC to real data over a period of time.

Example 1

Table 1 shows raw data collected during code peer reviews.

- Sample – series of peer reviews.
- Units – number of software units reviewed.
- SLOC – number of source lines of code reviewed.
- Defects – number of defects detected for each sample.
- Defects/1000 SLOC – defects normalized to 1000 SLOC for each sample.

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Feb 1997	17	1705	62	36.36
2. Mar 1997	18	1798	66	36.70
3. Mar 1997	15	1476	96	65.04
4. Mar 1997	19	1925	57	29.61
5. Mar 1997	17	1687	78	46.24
6. Apr 1997	18	1843	66	35.81
Totals	**104**	**10,434**	**425**	

Table 1. *Code Peer Review Defects*

The calculations are shown in Table 2.

The formulas for constructing the control chart follow [2]. The control chart used is a u chart.

- Defects/1000 SLOC = Number of Defects * 1000/SLOC reviewed per sample (calculated for each sample). These are plotted as Plot.
- Control Limit (CL) = total number of defects/total number of SLOC reviewed * 1000.
- A(1) = SLOC reviewed/1000 (calculated for each sample).
- Upper Control Limit (UCL) = CL+3(SQRT(CL/a(1)) (calculated for each sample).
- Lower Control Limit (LCL) = CL-3(SQRT(CL/a(1)) (calculated for each sample).

The control chart is shown in Figure 3.

Table 2. *Calculations for Code Peer Review Defects*

Sample	Plot	CL	UCL	LCL	A(1)
1. Feb 1997	36.4	40.73	55.4	26.09	1.7
2. Mar 1997	36.7	40.73	55.01	26.45	1.8
3. Mar 1997	65.0	40.73	56.49	24.97	1.5
4. Mar 1997	29.6	40.73	54.53	26.93	1.9
5. Mar 1997	45.2	40.73	55.47	25.99	1.7
6. Apr 1997	35.8	40.73	54.84	26.63	1.8

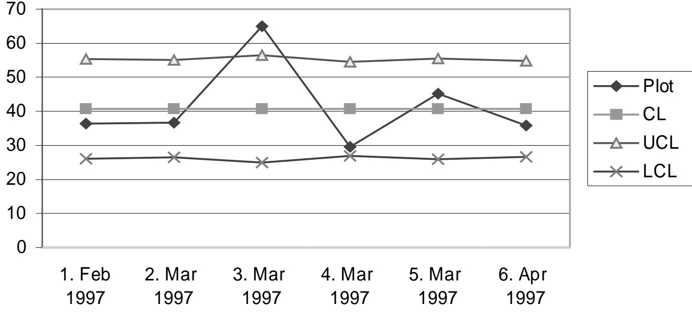


Figure 3. *Control Chart for Code Peer Review Defects*

The process is out of SPC in the third event. Causal analysis revealed this was caused when the project introduced coding standards and many coding violations were injected. The root cause is lack of knowledge of the coding standards. The defect prevention is to provide training whenever a new process or technology is introduced.

Example 2

Table 3 shows raw data collected at code peer reviews over a period of months:

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Mar 1998	6	515	15	29.12
2. Apr 1998	10	614	16	26.06
3. Apr 1998	7	573	7	12.22
4. Apr 1998	7	305	7	22.95
5. Apr 1998	4	350	21	60.0
6. Apr 1998	3	205	2	9.76
7. Apr 1998	8	701	11	15.69
8. May 1998	3	319	3	9.40
Totals	**76**	**3,582**	**72**	

Table 3. *Code Peer Review Defects*

Table 4 shows the calculations. LCL is set to zero when it is negative.

Sample	Plot	CL	UCL	LCL	a(1)
1. Mar 1998	29.13	20.1	38.84	1.36	0.515
2. Apr 1998	26.06	20.1	37.27	2.96	0.614
3. Apr 1998	12.22	20.1	37.87	2.33	0.573
4. Apr 1998	22.96	20.0	44.45	0	0.305
5. Apr 1998	60.00	20.1	42.84	0	0.35
6. Apr 1998	9.76	20.1	49.80	0	0.205
7. Apr 1998	15.71	20.1	36.16	4.04	0.701
8. May 1998	9.40	20.1	43.91	0	0.319

Table 4. *Calculations for Code Peer Review Defects*

The control chart is shown in Figure 4.

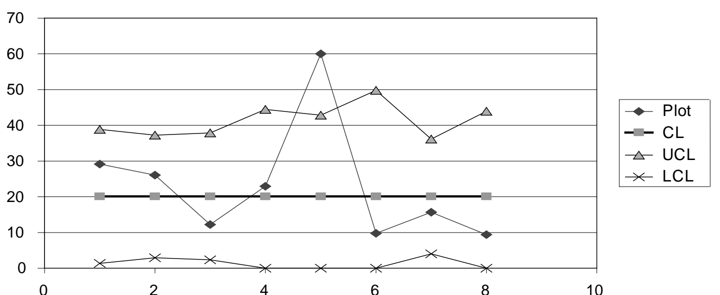


Figure 4. *Control Chart for Code Peer Review Defects*

An anomaly occurred in the fifth sample. Causal analysis revealed that data for that sample were for database code, all others were applications code. Control charts require similar data for similar processes, i.e., apples to apples analogy. The database sample was removed and the data charted again as shown in Figure 5.

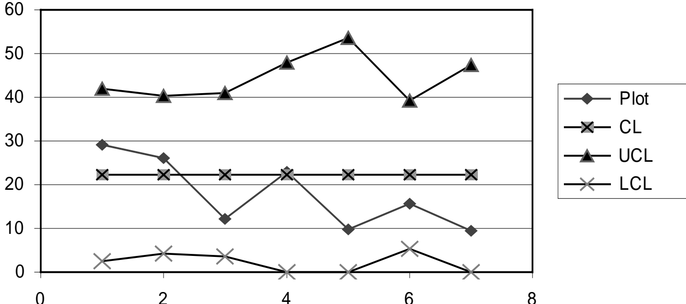


Figure 5. *Control Chart without Database Defects*

The process is now under SPC. The root cause is that data gathered from dissimilar activities cannot be used on the same statistical process on control charts. Data from design cannot be combined with data from coding. The process for database design and code is different from that used for applications design and code as are the teams and methodologies. The defect prevention is against the process of collecting data for SPC control charts.

Example 3

During integration testing, the defects were categorized against the test plan, test data, code logic, interfaces, standards, design, and requirements. Defects against these attributes are shown in Table 5.

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements
1	2	6					
2		10					
3	1	9	3				
4	2	1	13				
5		1	7				
6		10	14				
7		4	2				
8		28					
9			6				
10	1	3				2	
11		10					
12		9	1				
13		6	2	1			
14		5	7				
Totals	6	102	55	1		2	

Table 5. *Integration Test Defects*

Figure 6 plots the defects discovered during integration tests.

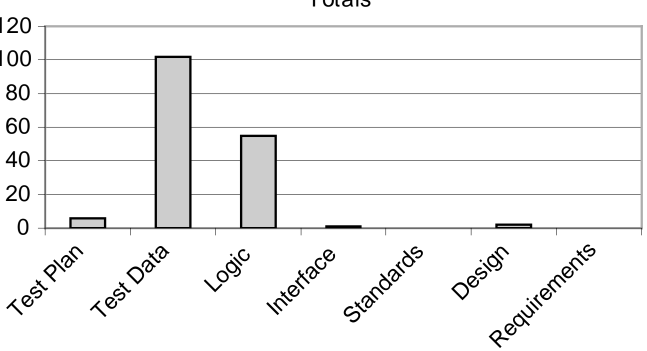


Figure 6. *Integration Test Defects Bar Chart*

Test data would not be expected to have the majority of defects. The root cause was that the test data in the test procedures had not been peer reviewed. The defect prevention mechanism is to peer review the test procedures and the test data.

Example 4

During preliminary design and prior to acquiring hardware, a simulated performance model was used to monitor critical computer resources. Figure 7 shows some results of monitoring the required MIPS. The model tracked estimated usage, the amount of MIPS required based on functional requirements to be implemented; availability, the amount of available MIPS in the model's design; and threshold, the number of MIPS that threaten the availability that requires remedial action.

Around November 1995, many new requirements were added to the system and the architecture's MIPS threshold was threatened because of increased computations. In May 1996, additional MIPS were added to the hardware design and the problem was corrected.

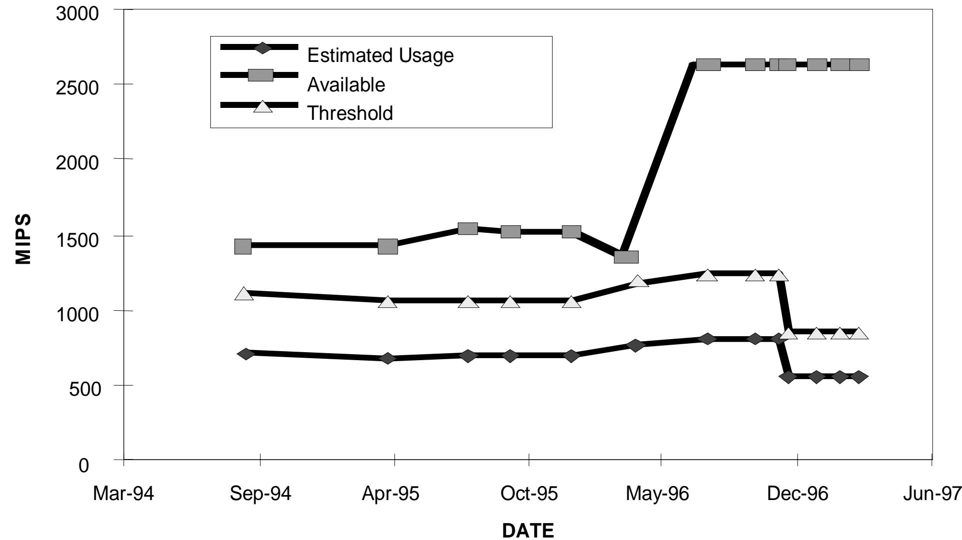


Figure 7. *General Purpose MIPS*

Conclusion

The use of rigorous statistics using SPC (control charts) and other statistical methods can easily and effectively be used in a software setting. SPC can identify undesirable trends and can point out fixable problems and potential process improvements and technology enhancements. Control charts can show the capability of the process, so achievable goals can be set. They can provide evidence of process stability, which can justify predicting process performance. SPC analysis can provide valuable information used in defect prevention and for lessons learned. SPC is new to software development but shows great promise that, in this author's opinion, will support process improvement, and will improve the productivity of development and the quality of products.◆

References

1. Paulk, Mark C.; Curtis, Bill; Chrissis, Mary Beth; Weber, Charles V., February 1993, *Capability Maturity Model for Software, V1.1,* Software Engineering Institute (SEI).
2. Florac, William A., Park, Robert E., Carleton, Anita D., *Practical Software Measurement: Measuring for Process Management and Improvement,* SEI, April, 1997.
3. Baumert, John H., McWhinney, Mark S., *Software Measures and the Capability Maturity Model*, SEI, 1992.
4. Florac, William A., Carleton, Anita D., *Measuring the Software Process, Statistical Process Control for Software Process Improvement,* SEI, 1999.
5. Radice, Ron, Getting to Level 4 in the CMM, 1997 SEI Software Engineering Process Group Conference, San Jose, Calif.

Suggested Readings

- Pyzdek, Thomas, *An SPC Primer*, Quality America Inc., 1994.
- Carleton, Anita; Paulk, Mark C.; Statistical Process Control for Software, 1997 Software Engineering Symposium, Pittsburgh, Pa.
- Chambers, David S.; Wheeler, Donald J., *Understanding Statistical Process Control*, SPC Press, 1995.
- *Juran's Quality Control Handbook*, 4th Edition, McGraw-Hill Book Company, 1988.
- Florence, Al, CMM Level 4 and Level 5 Plan, 1999 Software Engineering Process Group Conference Proceedings, Atlanta, Ga.
- Donald J. Wheeler, *Advanced Topics in Statistical Process Control*, SPC Press, 1995.
- Deming, W. Edwards, On Probability As a Basis For Action, *The American Statistician*, November 1975, Vol. 29, No. 4, pp.146-152.
- Florence, Al, CMM Level 4 Quantitative Analysis and Level 5 Defect Prevention, 2000 Software Technology Conference Proceedings, Salt Lake City, Utah.
- Humphrey, Watts S., *Managing the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Company, September 1997.

About the Author

Al Florence has worked at Hughes Aircraft, TRW, Martin Marietta, Science Applications International Corp., and currently at the MITRE Corporation. He has worked in all software life-cycle phases from concept to retirement in several disciplines, including systems, software, development, test, configuration management, and quality assurance, as both a developer and manager. He has diversified experience in real-time command and control, aircraft, spacecraft, missiles, weapon systems, particle accelerators, simulation, and information systems projects. He has developed processes for all CMM® key process areas at all CMM levels and is a trained evaluator and assessor. He has a bachelor's degree in mathematics and physics from the University of New Mexico and did graduate work in computer science at the University of California Los Angeles and the University of Southern California.

MITRE Corp.
1820 Dolley Madison Blvd.
McLean, Va. 22102-3481
Voice: 703-883-7476
Fax: 703-883-1889
E-mail: florence@mitre.org

Evolving Function Points

Lee Fischman
Galorath Inc.

Functional size metrics for software emerged a generation ago with the invention of the function point. Since then, they have become the most common alternative to lines of code. Function points gauge software size in terms of delivered functionality rather than gross physical size, providing a valuable alternative perspective that often is preferred. Despite being a key innovation in software sizing, the software engineering community has not been entirely satisfied with function points. Consequently, alternative functional metrics (Mark II, Feature Points, and Full Function Points) have been proposed to remedy perceived deficiencies. This diversity, however, does not lead the software community to a standard that achieves widespread use. Moreover as the leading functional metric, function points deserve to be evolved rather than abandoned. This article outlines the findings of a metrics research program conducted during the last several years. The program explored function points' underlying framework, reviewed previous research, and considered changes to the current standard. The goal is to reconcile lingering criticisms of function points with the tremendous investment made in them during the past 20 years.

What do critics claim is wrong with function points? The critique below may be a long list, but hold your breath. It is not damning. Function points have been shown to be a definite indicator of development effort, and are still fundamentally sound.

Semantically Difficult. Function point standards were codified in the early 1980s by a standards body hailing from a traditional management information system world. Since then the standards document has not been drastically overhauled. Its language reflects this with seemingly arcane terms such as "record element types, external inputs, etc." While such careful language insulates a relatively complex metric from everyday misunderstanding, it also impedes learning and acceptance by a wider audience.

Too Many Steps. The function point counting methodology is complex. It takes several days to learn function points, which is more time than most harried software engineers are willing to spend. Furthermore, some of that methodology is mathematically suspect while potentially adding no benefit.

Incomplete. Function points were defined from the user interface's vantage. Although a clever angle, this caused major criticism that all the functionality built into a software system might not be captured. Many argued that substantially internal functionality, without much manifestation at the user interface, might be missed.

Arbitrary Weightings. Once identified, *raw* function points go through two numeric transformations. The first is meant to weight them for relative size—low, average, high. The second is intended to make different types of points comparable such as equating an external input to an external output. The problem is that the scalar values behind these transformations were developed more than 20 years ago under very particular circumstances. At worst, these values may now be arbitrary.

No Automatic Count. No generally automated method is available for counting function points, even in completed systems. In contrast, lines of code counts can be obtained using simple line counting utilities. This paper does not address the automatic counting issue; innovations eventually may emerge from computer aided software engineering vendors.

Simple Semantic Changes

The following changes are intended to make function points easier to learn and eliminate inconsistencies.

Simpler Names. Function points' key innovation is that they approach software size from an intuitive perspective—user interface artifacts such as inputs, outputs, and files a software developer understands. Why call these *external inputs, external outputs and internal logical files* when more straightforward terms work equally well? Figure 1 offers a simplified nomenclature.

Figure 1. *Simplified Naming Scheme*

External Input	External Output	External Inquiry	Internal Logical File	External Interface File
⬇	⬇	⬇	⬇	⬇
Input	Output	Simple Input/Output	Internal Data Grouping	External Data Grouping

Simplified Weighting Terms. The function point methodology describes a function in terms of size. Actually, the *standard* refers to "complexity" but complexity is an algorithmic factor that should be orthogonal to a size metric, so we are unilaterally changing the label. Consistent with this change, low, average, and high complexity become *small, medium,* and *large.*

Size is determined by counting a function's attributes. The standard refers to these as data element types, record element types, and file types referenced—simpler terms are *field* for the first item and *data groupings accessed* for the latter two. Figure 2 illustrates how size is determined then labeled using the alternative nomenclature outlined here.

Accounting for Hidden Functionality

Function points are determined at an application's external

Figure 2. *Size Determination Matrix*

		Total Number of Fields		
		1 to 5	6 to 19	20 or more
Data Groupings Accessed	1	Small	Small	Medium
	2 to 3	Small	Medium	Large
	4 or more	Medium	Large	Large

interface, the layer where interaction with the outside world occurs. However, attributes at the external interface sometimes provide little indication of how substantial underlying code is. Examples include algorithmically intense software (encryption, image processing) or systems with underlying "layers" that are out of the user's view. Judged from the external interface, the size of these systems will be understated. Two very different methods for capturing hidden size have been suggested but never before specified for use in a single framework.

An Internal Function Point. Numerous researchers have suggested a new function point to capture functionality missed by the other categories. It even has been implemented in competing functional metrics schemes. Figure 3 illustrates the idea behind the "internal function."

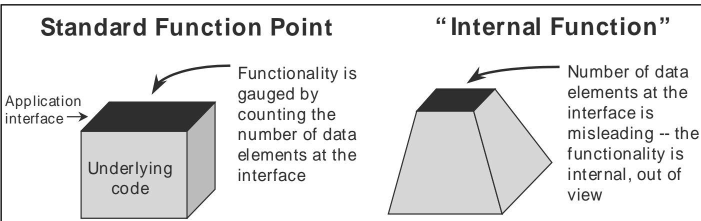


Figure 3. *The Internal Function "Iceberg"*

As depicted an internal function is a truly extraordinary input or output. It easily bests other functions that form an external perspective resembling it in size but have nowhere near the underlying amount of code. Keep in mind that these functions should occur rarely, no more than a few times in the average system.

When an internal function is found, it probably should be sized by analogy against standard function points. Compare an internal function against other known inputs or outputs in the system—it could equal *several*. Remember that an internal function is an input or output with a misleadingly simple external interface; sizing by analogy corrects this misjudgment.

Layers. Other hidden functionality can be captured by a change of perspective. A cornerstone of the function point framework is that software functionality, except key data structures, is not functional unless it interacts with the outside world. This external interface provides a consistent vantage while accounting for the entire system. However, this level can also conceal the inner workings of a complex system (see Figure 4).

Component-to-component interaction can be revealed with internal layers, an innovation first proposed for full function points. Beneath the external interface, layers are intended as equally valid perspectives from which to count function points. To prevent misinterpretation and overcounting, a layer must be strictly defined. All software has many functions interacting with one another; these do not justify layers.

Internal layers are characterized by a well defined internal interface that every function in a system lies either above or below. They are tantamount to secondary application boundaries. Layers certainly exist when there are wholly constituted systems within systems, such as with middle-ware and operating system utilities. Inputs or outputs counted at each layer still must satisfy the counting rule that an internal file (data grouping) is modified.

Figure 4. *Layers*

External Interface

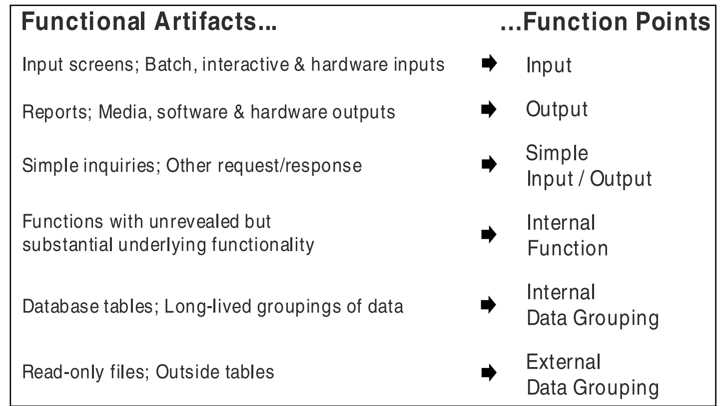
Application

lower layer applications interact only *within* system

Methodology Changes that Aid Learning

It takes several days to learn function point counting and more time to become proficient. This hurdle has limited the pool of trained counters. However, an exploration of the standard reveals potentially easier ways to learn to count.

Start from Artifacts. An alternative approach to learning function points is to start with a set of recognizable design artifacts and provide clarification only when necessary. This should be a much faster way to learn that establishes a critical intuitive link for skeptical software developers, the target audience. Figure 5 suggests mappings between artifacts and function points.



Functional Artifacts...		...Function Points
Input screens; Batch, interactive & hardware inputs	➡	Input
Reports; Media, software & hardware outputs	➡	Output
Simple inquiries; Other request/response	➡	Simple Input / Output
Functions with unrevealed but substantial underlying functionality	➡	Internal Function
Database tables; Long-lived groupings of data	➡	Internal Data Grouping
Read-only files; Outside tables	➡	External Data Grouping

Figure 5. *Mapping from artifacts*

Counting Rules Only as a Last Resort. Judging function points from artifacts is a shortcut that every experienced counter takes. However, a function is not a "point" unless specific rules are satisfied. These reinforce the formal framework behind function points and help to resolve discrepancies. The rules have been reformulated to make them slightly easier; these are not currently endorsed by any formal standards organization. The counting rules for transactional functions (inputs, outputs, input/output) can be reduced to three core rules that establish which observations constitute a point:

- Does this process leave the system in an equilibrium state? From the user's perspective, this means that nothing is left to be done. The entire sequence of actions until a feature is satisfied should be considered part of a single point.
- Is this the smallest meaningful unit of activity? If adjacent pieces of functionality can work separately and each satisfy discrete functional requirements, then count them separately.
- Is the logic or data being handled unique to this process? If not unique, this functionality should not be counted.

The counting rules for files, recast in this article as data groupings, are:

- Is this group of data visible to the user via an input or output? Groupings of data are evaluated at the external interface

or internal layer (if you can accept the latter as an extension) and so they must naturally be evident there.
- Does this group of data logically belong together? If certain data items are always associated, then they belong in a single group. This reinforces the idea that function points are based on specifics of design rather than implementation. As such, physical attributes (tables, flat files, etc.) can delineate logical groupings of data.
- Has this group of data been counted before? A data grouping may be encountered in a system many times, but it only is designed (and counted) once.

The Math

Alongside the qualitative definition of function points there is a mathematical framework that is necessary for quantifying and summarizing them. Function points can be used for quantitative purposes such as for effort estimation only after they are transformed into a numeric value such as in effort estimation. Yet the standard methodology involves a loss in information and may be somewhat arbitrary.

Do not summarize into unadjusted function points. A crucial step in orthodox function point analysis is taking separately counted inputs, outputs, files, etc., and combining these into a single value, the *unadjusted function point count*. However, whether a function point is a file, input, or output is important information that is lost when function points are rolled into a single value. Function point counts by type should be retained so a maximum amount of information is available for later use.

If you are going to use weightings, be careful. Function points are counted by type and then weighted by size (see Figure 2). However, the weighting factors in common use were determined from IBM applications in the late 1970s. There is no proof they can be generalized to other organizations, technologies, and eras.

A few things can be done. First, accept the weightings. There is no alternative to them, and the standard weightings are required for comparison against third-party actuals databases. Alternatively, ignore the weightings and simply count by type, ignoring size. This approach could work if the function point count is used only for in-house purposes. A final option is to develop your own weighting scheme, perhaps backed up by another metric or by known effort relationships.

Conclusion

If every recommendation in this article were to be adopted, the result would be a function point standard that is markedly similar to the current one. The various simplifications proposed do not change counting results; meanwhile, extensions to account for hidden functionality would only rarely apply. Other suggestions are intended to increase the acceptance of function points and involve no changes to the underlying standard.

Functional size metrics are here to stay. As software technology continues to evolve, they eventually may be preferred to lines of code. The question is whether lingering concerns about function points will remain unanswered or whether many of the changes advocated here will be adopted.

Acknowledgments

Thanks to Alan Clark for editing assistance.

Epilogue

For a further understanding of function points, go to www.galorath.com/fp_tutorial. The internal function defined here is different from those previously proposed in that it must manifest at the user interface. The reason for this difference is the simultaneous provision for internal layers, which should capture truly internal functionality. If you have better ideas on how to size internal functions or how to transform a qualitative size scale to a numeric value, contact me.◆

Recommended Readings

- Abran, Alain and Robillard, Pierre N., Function Points Analysis: An Empirical Study of Its Measurement Processes, *IEEE Transactions on Software Engineering,* Vol. 22, No. 12, pp. 895-910, December 1996.
- Albrecht, Allan J., Gaffney, John E., Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, *IEEE Transactions on Software Engineering,* Vol. SE-9, No. 6, pp. 639-648, Nov. 1983.
- Bock, Douglas B., Klepper, Robert, FP-S: A Simplified Function Point Count Method, *The Journal of Systems and Software,* July 1992, Vol. 18, No. 3, pp. 245-254.
- Briand, Lionel, El Emam, Khaled, and Morasca, Sandro *Theoretical and Empirical Validation of Software Product Measures,* Technical Report, Centre de Recherche Informatique de Montréal, Number ISERN-95-03, 1995.
- Fischman, Lee, *Analysis Of Function Point Rules In A Tree,* presented at the 1999 International Workshop On Software Metrics, available at www.galorath.com
- Fischman, Lee, *The Place of Function Points In An Underlying Model of Software Content,* presented at the 1999 IFPUG National Conference, available at www.galorath.com
- Fischman, Lee, Function Point Counting For Mere Mortals, presented at the 1999 Applications of Software Metrics Conference, available at www.galorath.com
- Harrison, Warren and Miluk, Gene. The Impact of Within Size Variability on Software Sizing Models, unpublished, available at www.galorath.com
- Ho, V.T., Abran, A., Oligny, S., Using COSMIC-FFP to Quantify Functional Reuse in Software Development, Escom-Scope 2000, available at www.lrgl.uqam.ca/ffp.html
- International Function Points User Group, *IFPUG Counting Practices Manual,* Version 4.1, www.ifpug.org
- Jones, Capers, Feature Points (Function Point Logic for Real Time and System Software), presented at the fall 1988 IFPUG National Conference.
- Kemerer, Chris F., Reliability of Function Points Measurement: A Field Experiment, MIT Sloan School of Management. WP#3193-90-MSA.
- Kitchenham, Barbara and Pfleeger, Shari L., and Fenton, Norman Towards a Framework for Software Measurement Validation, *IEEE Transactions on Software Engineering,* 21(12), pp. 929-943, December 1995.
- Oligny, Serge and Abran, Alain, On the Compatibility Between Full Function Points and IFPUG Function Points, unpublished, available at www.uqam.ca
- Symons, Charles R., *Software Sizing and Estimating Mk II Function Point Analysis,* John Wiley & Sons, 1991.

About the Author

Lee Fischman is a special projects director at Galorath Incorporated and is responsible for applications development, design, and research projects. Fischman is also a frequent speaker at national conferences. He studied economics at the University of Chicago and UCLA.

Galorath Incorporated
100 North Sepulveda Boulevard, Suite 1801
El Segundo. Calif. 90245
Voice: 310-414-3222
Fax: 310-414-3220
E-mail: info@galorath.com

QUOTE MARKS

Software is like Entropy: It's hard to grasp, weighs nothing, and obeys the second law of thermodynamics, i.e. it always increases.

— Norman Augustine

"A language that doesn't affect the way you think about programming is not worth knowing."

— Anonymous

"Man invented language to satisfy his deep need to complain."

— Lily Tomlin

Coming Events

February 7-9
Network and Distributed System Security Symposium
www.isoc.org/ndss01/call-for-papers.html

February 12-16
Software Management Conference
www.sqe.com/sm

February 12-16
Applications of Software Measurement Conference
www.sqe.com/asm

March 5-8
Software Testing Analysis and Review
www.sqe.com/stareast

March 5-8
Mensch and Computer 2001
http://mc2001.informatik.uni-hamburg.de

March 12-15

Software Engineering Process Group Conference
www.sei.cmu.edu/products/events/sepg

March 31-April 5
Conference on Human Factors in Computing Systems
www.acm.org/sigs/sigchi/chi2001

April 22-26

Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies
www.ieee-infocom.org/2001

April 29-May 4

Software Technology Conference (STC 2001)
www.stc-online.org

May 1-3
2001 IEEE Radar Conference
www.atlaessgrss.org/radarcon2001

May 12-19
23rd International Conference on Software Engineering, and *International Workshop on Program Comprehension*
www.csr.uvic.ca/icse2001

June 03-06
2001 IEEE Southwest Test Workshop
E-mail: william.mann@ieee.org

June 11-13
E-Business Quality Applications Conference
http://qaiusa.com/conferences/june2001/index.html

June 18-22
ACM/IEEE Design Automation Conference
www.dac.com

June 25-27
2001 American Control Conference
www.ece.cmu.edu/~acc2001

STC 2001

THE THIRTEENTH ANNUAL

Software Technology Conference

2001 Software Odyssey: Controlling Cost, Schedule, and Quality

APRIL 29 – MAY 4, 2001 • SALT LAKE CITY, UTAH

How far has technology come since the dawn of the computer age? Have we reached the perfection of intelligence demonstrated by HAL 9000 in 2001: A Space Odyssey*? After all, HAL never committed a recordable error. We continue this fantastic journey, searching the depths of software and systems technology for solutions to controlling cost, schedule, and quality. Join us in Salt Lake City, in the real year 2001, as today's leading technology experts look toward the future and explore opportunities for perfecting today's technology.*

Sponsors

The Software Technology Conference (STC) is co-sponsored by the Departments of the Army, Navy, and Air Force, the Defense Information Systems Agency (DISA), and Utah State University Extension. In its thirteenth year, STC is the premier software technology conference. We anticipate more than 3,000 participants this year from the military services, government agencies, defense contractors, industry, and academia.

The government co-sponsors are **Lt. Gen. Harry D. Raduege Jr.**, director, DISA; **Lt. Gen. Peter M. Cuviello**, director of Information Systems for Command, Control, Communications, and Computers, U.S. Army; **Rear Adm. Kenneth D. Slaght**, vice commander, Space and Naval Warfare Systems Command, U.S. Navy; and **Dr. Donald C. Daniel**, deputy assistant secretary of the Air Force for Science, Technology, and Engineering, U.S. Air Force.

General Sessions, Panel Discussion, Plenary Speakers

The general session will be held Monday afternoon and features keynote speaker **Dr. Vitalij Garber**, director of Interoperability, Under Secretary of Defense (Acquisition, Technology, and Logistics). The co-sponsors will host a panel discussion Tuesday morning, moderated by **Dawn C. Meyerriecks**. Wednesday and Thursday mornings will begin with plenary speaker sessions, with **Steven R. Perkins**, senior vice president and general manager of Oracle's Federal and Global Financial Services, as Wednesday's featured speaker, and **Dr. Eliyahu M. Goldratt**, educator and creator of the Theory of Constraints, as Thursday's featured speaker. **Gen. Lester Lyles**, commander, Air Force Materiel Command, has been invited to speak at Thursday afternoon's closing general session with **Maj. Gen. John L. Barry**, director of Strategic Planning, deputy chief of staff for Plans and Programs, Headquarters U.S. Air Force.

Special Sessions

Sponsored track presentations will be offered throughout the week by the following organizations:

Navy CIO: *Data Management and Interoperability.*
Air Force: *DII COE Real-Time Extensions– A Year's Worth of Accomplishments for Warfighter Platform Development.*
OSD: *DoD Software Intensive Systems Initiative.*
GSA: *Federal IT Accessibility Initiative – Section 508.*
CAC: *Common Access Card (Smart Card).*
INCOSE: *Supporting, Simplifying, and Streamlining the Odyssey.*
STSC: *Theory of Constraints.*
IEEE: *Putting IEEE Best Practices to Work Today for DoD Software Engineering Success.*
DACS: *Intelligent Software Agents.*

Federal IT Accessibility Initiative – Section 508

How does Section 508 impact your organization? Section 508 is a portion of the Rehabilitation Act that requires access for disabled persons to the federal government's electronic and information technology. We are excited to partner with the U.S. General Services Administration (GSA) in offering a Monday tutorial and a sponsored track, including a panel discussion, to aid in educating our participants on the new standards set forth by Section 508. These sessions will address issues such as Webmaster training, procurement guidelines, legal and budgetary implications, team building, and strategic planning. In addition, GSA will host their Cyber Café in the exhibit hall, which will showcase assistive technology through actual hands-on demonstrations. For more information on Section 508, visit their Web site at www.section508.gov

On the Agenda

Presentation track topics include, but are not limited to:

- Assessments and Evaluations
- CMM®
- CMMI®
- Collaborative Engineering
- Configuration Management
- COTS
- Critical Systems
- Data Management – Mining/Warehousing/Sharing
- DII COE
- Distributed Computing
- Earned Value
- E-Commerce
- Education and Training
- Embedded Software and Systems
- Emerging Technologies
- Information Assurance
- Internet/Intranet
- Interoperability
- Knowledge Management
- Measurement
- Middleware
- Network Security
- Object-Oriented Technology and Languages
- Open Systems
- Outsourcing and Privatization
- Project Manager Training
- Process Improvement
- Program Management – Postmortem/Methodologies/Certification
- Quality Assurance
- Risk Management
- Simulation-Based Acquisition
- Software Acquisition
- Software Architecture
- Software Estimation
- Software Implementation
- Software Policies and Standards
- Software Sustainment
- Software Testing
- System Acquisition
- System Requirements
- Web-Based Solutions
- XML

STC 2001 Exhibiting Organizations (As of 11/30/00)

Ada Core Technologies Inc.
Air Mobility Command Computer Systems Squadron
Amdahl Software
AP Labs
BEA Systems Inc.
BMC Software Inc.
Boeing Co.
CDW Government Inc.
Cognos Corp.
Computer Resources Support Improvement Program Legacy Software Support
DCS Corp.
DDC-I
DISA
EDS
Federal Data Corp.
Fuentez Systems Concepts
Galorath Inc.
Harris Corp.
Health Information Resources Service
IBM Corp.
Integrated System Diagnostics Inc.
Jacada Inc.
Lockheed Martin
Logicon
Lotus Development Corp.
MapInfo Corp.
MARCORSYSCOM
MERANT
Novell Inc.
NXi Communications Inc.
Objective Interface Systems Inc.
OO-ALC/TIS
Open Systems Joint Task Force
PeopleSoft Inc.
Platform Computing Corp.
pragma SYSTEMS CORP.
Praxis Critical Systems
Predicate Logic Inc.
QSM Inc.
Quality Plus Technologies Inc.
Rational Software
Real-Time Innovations Inc.
RS Information Systems Inc.
RSA Security Inc.
Science Applications Intl. Corp. (SAIC)
Scitor Corp.
Section 508 / General Services Administration
SilverStream Software Inc.
Software Configuration Solutions Inc.
Software Engineering Institute
Software Productivity Consortium
Software Technology Support Center
SPAWAR
SSG/MSG
TeraQuest
Tivoli Systems
Tumbleweed Communications Corp.
United Defense
U.S. Air Force
U.S. Army
U.S. Army Information Systems Engineering Command
U.S. Army Strategic and Advanced Computing Center
USAA
USACECOM
Utah State University Extension
Vitech Corp.
WesTest Engineering Corp
WR-ALC/ LYS Software Engineering Division

Navy CIO – Data Management and Interoperability

Getting the most from our information technology infrastructure requires effective data management. The **Data Management** portion of this track will provide attendees an opportunity to hear about data management initiatives within the DoD that are focused on improving interoperability and efficient operations. Interoperability implies the existence of diverse systems that need to exchange data and services. The **interoperability** portion of this track will focus on interoperability across each phase of the software engineering life cycle.

Common Access Card

In October 2000, the DoD began issuing smart cards as the newest and most technologically advanced type of identification card, the DoD Common Access Card (CAC). The CAC will be the standard identification card for approximately four million active duty uniformed services personnel, selected reserve, DoD civilian employees and eligible contractor personnel. It offers the same benefits and privileges as the current identification card and will also be the principal card used to enable physical access to the department's buildings and controlled spaces or gain access to the department's computer networks and systems. Tuesday's track six presentations focus on this new identification card.

Networking Events, Optional Activities

STC 2001 features daily networking opportunities with the Opening Welcome Reception Monday, Tuesday night's optional Salt Lake City Overview tour, Wednesday's "Drag 'n Drop" Social, and "1964"… The Tribute—an evening of entertainment, the "Light Byte" luncheon in the exhibit hall Thursday, and the optional dinner cruise that evening. Space for these events is limited; companion packages are available.

Registration

Completed registration form and payment must be received by March 26, 2001 to take advantage of the early registration fees. Credit cards will not be charged until April 2, 2001. The conference fee structure for STC 2001 is as follows:

Discounted registration fee (paid by March 26, 2001):

Active Duty Military/Government*	$560
Business/Industry/Other	$685

Regular registration fee (paid after March 26, 2001):

Active Duty Military/Government*	$620
Business/Industry/Other	$770

* *Military rank (active duty) or government GS rating or equivalent is required to qualify for these rates.*

Housing Reservations

Housing reservations are handled by the Housing Bureau of the Salt Lake Convention and Visitors Bureau using the on-line Passkey system. Housing has been available since May 2000, therefore some government rate guestrooms at specific hotels may not be available. To access the Passkey system, log on to the STC Web site at www.stc-online.org and select the Housing reservation button. You will receive immediate "real-time" confirmation of your reservation. If you prefer to make your reservation using a traditional method, a PDF version of the housing form is available on-line.

Trade Show

STC 2001 will again feature its accompanying trade show, providing more than 180 exhibitors the opportunity to showcase the latest in software and systems technology products and services. This year's schedule has been adjusted to allow participants more time to interact with the vendors without conflicting with conference presentations.

Exhibit space is sold in increments of 10' x 10' at a rate of $1,575 per 10' x 10' space if application is received on or before February 16, 2001. Should space still be available after this date, booth space shall be processed at the rental rate of $1,775 per 10' x 10' space. Special fees and restrictions may apply to certain types of booth space. Complete trade show rules, regulations, and updated hall layout are available on the STC Web site.

New this Year! All badged exhibit personnel wishing to attend the entire conference are eligible for a discounted conference registration fee. Please utilize the conference registration form that was mailed to the exhibit manager in early January to register for the full conference.

www.stc-online.org

General Information
stcinfo@ext.usu.edu
435-797-0423

Technical Content Inquiries
stc@hill.af.mil
801-777-7411

Trade Show Inquiries
stcexhibits@ext.usu.edu
435-797-0047

Media Relations
stcmedia@ext.usu.edu
435-797-1349

Definition of Terms, continued from page 15

Function Point. One standard unit of delivered or finished software size, analogous to a gallon of milk, a case of beer, or a cord of wood**.** The size of a software package, from the viewpoint of a user, is its number of function points. A function point is unadjusted until it is weighted according to the overall application value adjustment factor. When using the term function point, it is usually understood that it refers to the adjusted or final function point. The textbook definition describes it as *"A metric that describes a unit of work product suitable for quantifying application software."*

General Systems Characteristics (GSCs). GSCs are 14 additional factors used to determine size/complexity of software. These include the degree of importance of such factors as reusable code, on-line data entry, and complex processing. These are used to increase or decrease the unadjusted function points by a value adjustment factor of up to +/- 35 percent.

Graphical Method. This method of solving certain small linear programs requires the analyst to plot the available resource equations on a graph. Then the analyst finds the point of intersection of two of the equations that represents the highest degree of achievement of the stated goal.

Internal Logical File (ILF). The ILF is a database that is inside the application. An ILF has seven, 10, or 15 unadjusted function points depending on whether it is of low, average, or high size/complexity. The textbook definition includes *"... a user identifiable group of logically related data or control information maintained within the boundary of the application."*

Linear Programming. Linear programming is an approach to solving problems using specially designed algorithms. One or more of these algorithms are usually taught in graduate schools of business. Two such algorithms are the "graphical method," and "Simplex."

Objective Function. The objective function is the goal to be reached, to the best possible degree, using linear programming. This goal is expressed in mathematical terms.

Primal. This is a certain perspective of defining the resources available to reach the stated objective in linear programming.

Record Element Type (RET). An RET is a logical subgroup type of data within a database, also defined as *"User recognizable subgroups of data elements within an ILF or EIF"*

Super File. A super file is a database (ILF or EIF) which contains more than 100 DETs. This is defined in CPM 3.4. According to the super file rule (SFR), if an ILF or an EIF contains more than 100 DETs, each RET is considered a unique ILF or EIF and is counted as such. This SFR is not currently recognized by IFPUG, although it was previously recognized in CPM 3.4.

Redundant Constraint. It is theoretically possible to formulate some algorithms with equations that do not affect their solutions. The algorithm correctly executes, but does not need the extraneous equation(s). This type of extraneous equation is called a redundant constraint. Since it does not contribute to the functionality of the algorithm, it is not included in the algorithm's function point count.

Simplex. Simplex is an algorithm for solving linear programming problems. George Dantzig developed it in 1947. In principle, Simplex is executed by first expressing the goal of a problem in mathematical terms (the objective function). Then, the resources available to use to reach the goal are expressed in terms of equations. By solving these equations simultaneously in a certain fashion, one calculates the best possible degree of achievement of the goal and the resulting blend of the resources needed. See [2] for an example of a text that treats linear programming.

Unadjusted Function Point. This is the size and complexity of a unit of software, before considering the effect of the value adjustment factor (VAF). For example, an EI that updates seven data fields in one ILF has the size and complexity of three unadjusted function points. When the effect of the VAF is considered, the three function points can be adjusted upward or downward by up to 35 percent depending on the degree of GSC contribution.

Value Adjustment Factor (VAF). The VAF is the computed variable used to convert an *unadjusted* function point to an *adjusted* or *final* function point. It is computed from the contribution to the software requirements of the GSCs. Using algebra, the VAF can be shown to have values from 0.65 to 1.35.

Get Your Free Subscription

Fill out and send us this form.

OO-ALC/TISE

5851 F Ave., Bldg 849, Rm B-04

Hill AFB, UT 84056-5713

Attn: Heather Winward

Fax: 801-777-5633 DSN: 777-5633

Voice: 801-586-0095 DSN: 586-0095

Or use our online request form at www.stsc.hill.af.mil

NAME:________________________

RANK/GRADE:__________________

POSITION/TITLE:_______________

ORGANIZATION/COMPANY:__________________

ADDRESS:______________________

____

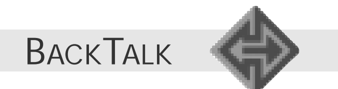
BASE/CITY:____________________

STATE:_____ ZIP:__________

VOICE: __________________

FAX:____________________

E-MAIL: ____________@_____________



Feng Shui for Not-Really-Dummies, aka Engineers

Feng shui [pronounced *fung shway*] is the ancient art of situating or orienting objects to promote a healthy flow of *qi* (vital energy, pronounced *key*). Its postulate is that all areas, large and small, have a distinctive energy that is guidable by rearranging objects [1]. Feng shui is not chop suey, and there is no corresponding number on the menu to use if you cannot figure it out.

Describing feng shui (FS) to an engineer can be like … describing FS to an engineer. It helps to incorporate acronyms. Also a non sequitur figure dropped into the article with little to no explanation (see Figure 1) can help them stroke qi instead of keys.

I found myself consulting FS experts after our office was recently remodeled. Everyone was happy with the new systems furniture except for me. Something was not quite right. I had been placed with my back directly opposite the door, and I felt drained, edgy, and irritable. My qi was being replaced by *sha,* or disruptive, negative energy.

Richard Craze and Roni Jay say "It is considered bad feng shui to sit with your back to a door. In ancient China this was considered an ill omen and you were bound to suffer a *loss of face* … if you sit with your back to a door you will feel uncomfortable as you can't see/know … what's going on behind you. That's common sense—but it is also feng shui [2]."

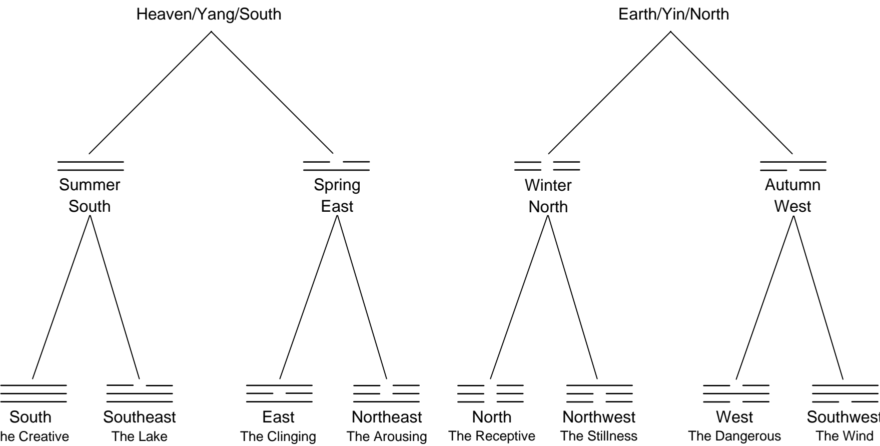


Figure 1. *The Eight Trigrams*

Oh oh, they said *common sense (CS).* As we read in a previous BACKTALK [D. Cook, June '00], some engineers don't know CS from a hole in the ground.

Craze and Jay continue, "Feng shui is about putting practical common sense into a clear and coherent form so we can all benefit." According to FS, a dripping faucet indicates that one is squandering wealth, allowing it all to wash down the drain. According to engineers, a dripping faucet indicates poor plumbing. Per FS, one should never leave the seat up while flushing. Most engineers have already heard this maxim, and are probably more concerned with whether the TP spools from the top or the bottom of the roll.

FS states that the ideal house should face south and be situated alongside running water or a winding (not straight) road. The eight trigrams come together at compass points to form the *bagua* (pronounced pah kwa), which may be placed over any house or room to diagnose and remedy FS. Have I lost you yet? Come on engineers—it's geometry! [See Figure 2.] The fame enrichment is always placed over your front door, whether it faces south or not. If your door faces a different direction or your house is an odd shape, you simply realign/stretch the bagua accordingly. See Table 1 for the eight enrichments and remedies.

When I applied the bagua to the floorplan of our new systems-furnished office, which faces north and thus is like a mirror [which you shouldn't rush out to buy unless you are told to do so by an FS consultant, your spouse, *and* an interior decorator (note may be out of sequence; I love this lengthy sentence)] image of the one you see to your right, I discovered that the area had neither a *health* nor a *pleasure* octant. Imagine that … and I was still sitting with my back to the door.

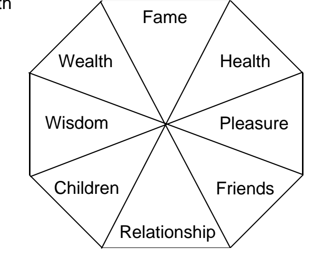


Figure 2. *The South-Facing Bagua*

So I asked my boss, who ironically sits in the *wealth* octant, to go to the FS shop and buy me a remedy. She came back with a lovely red, black, and gold bagua straight-lines remedy with a mirror in the middle to bounce the bad behind-my-back qi out the door from whence it came.

Oh by the way, FS is based on the *Wu Hsing* (not a condiment), or Theory of Five Aspects, which states that we are all a combination of elements—wood, fire, metal, water, and earth—with one of them in a position of dominance determined by the year we were born. I am a metal dog. This is not a place mat. Get it?

Table 1. *FS Enrichments/Remedies*

Enrichment	Remedy	Examples
Fame	Light	Mirrors or reflective surfaces.
Health	Straight Lines	Flutes, swords, fans, scrolls, etc.
Pleasure	Stillness	Statue or a large rock.
Friendship	Sound	Wind chimes or bells; also fountains.
Relationship	Movement	Flags, banners, mobiles, fountains.
Children	Color	Lucky colors: red, white, gold, black.
Wisdom	Mechanical Device	Engineers' complete toy set …
Wealth	Living Things	Plants or fish.

– Matt Welker, Shim Enterprise Inc., has read two books on feng shui and therefore considers himself an expert.

References

1. *C-Health's Alternative Health Dictionary* [www.canoe.ca/AltmedDictionary].
2. Craze, R. and Jay, R., *Teach Yourself Feng Shui.* Hodder and Staughton, London, England, 1998.

FS myths are debunked at www.qi-whiz.com

STC 2001

THE THIRTEENTH ANNUAL

Software Technology Conference

2001 Software Odyssey: Controlling Cost, Schedule, and Quality

The Premier Software Technology Conference

29 April - 4 May 2001
Salt Lake City, Utah

Complete Conference Information and Registration Available at

www.stc-online.org

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
5851 F Avenue
Building 849, Room B04
Hill AFB, UT 84056-5713

PRSRT STD
U.S. POSTAGE PAID
Kansas City, MO
Permit 34