

Surfing the Net for Software Engineering Notes

Mark Doernhoefer

The MITRE Corporation
7515 Colshire Dr.
McLean, VA 22102
mdoernho@acm.org

Solving the Software Quality Problem

The subtitle to this month's article is just meant to grab your attention. I assure you, this column will **not** be solving the software quality problem. What I do hope to do is point you at some interesting web sites that discuss potential solutions to the software quality problem and let you solve it on your own.

The issue of improving the state of the art of software engineering has been roundly debated for years. Some believe the best way to improve the quality of software is by licensing programmers and software engineers. Others argue that software quality will only be improved if we improve the educational foundations of the practice; do a better job of teaching software engineering in our colleges and universities. Still others propose techniques and tools as the answer to the quality problem.

The debate has been long and, at times, loud. I will try to stay neutral in my presentation of the various points of view and let you decide. I recommend you download an electronic version of this column and surf by the links presented here. You don't have to read the entire content of each site, but you'll want to scan the sites to give you some food for thought.

There is a wide range of opinion as to what actually constitutes "quality" as it applies to software. For many, the definition of quality software is simply bug-free code. However for those tasked with maintaining a software product, the definition of quality goes much deeper to address ease of maintenance; degree to which the software can be adapted or extended; the ability of the code to handle odd or unusual inputs; or other factors. Wikipedia, in its article on the Software Quality Model defines software quality as:

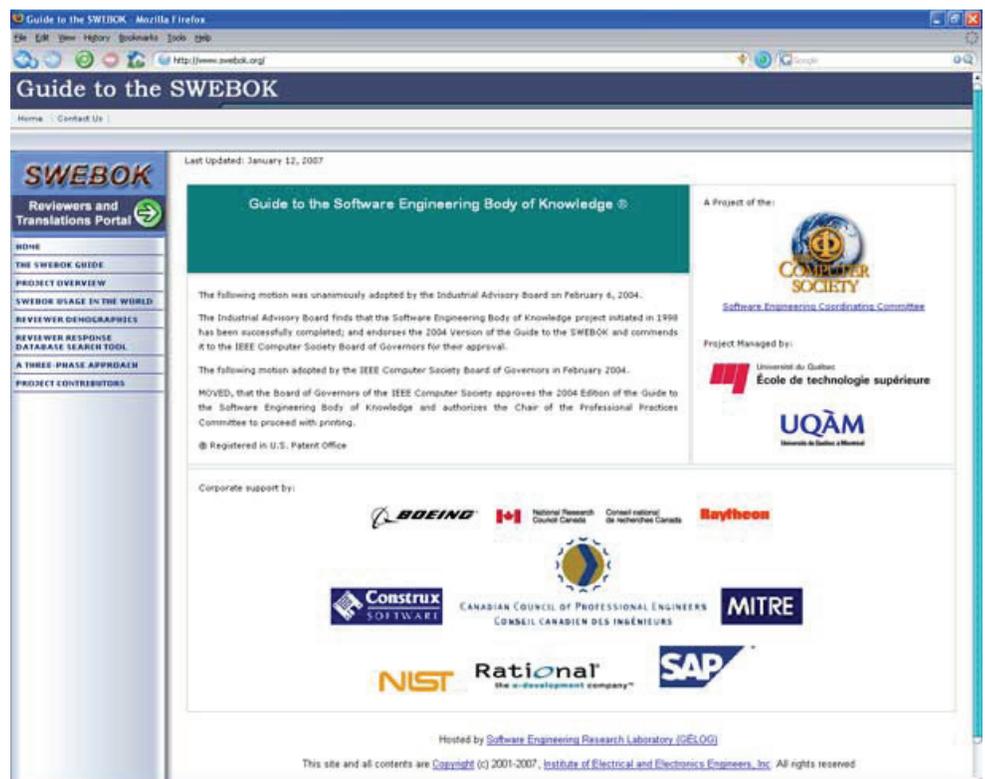
"Software quality can be defined as 'conformance to requirements' and/or 'fitness of use'. Quality

achievements start with a loud and clear definition of what "quality of source code" means to your organization or project. In simple terms all the stakeholders must be well informed of what is expected, what are the goals to be achieved, what is evaluation criteria and how they can contribute to achieve the goal."

At a high level, that is a definition of quality shared by many program and project managers. How about something more formal?

There is an ISO standard, ISO/IEC 9126, which attempts to formalize the definition of software quality by establishing standards for modeling and measuring quality. The standard, in four parts, describes a software quality model, external metrics for measuring quality, internal metrics for measuring quality and a set of quality in use metrics. The ISO standards are copyrighted and not available on the web. You can purchase the standards set for about US\$500 from the ISO website at: <http://www.iso.org/>.

The sites discussed in this column will address many of the quality aspects of ISO 9126. You can apply education, tools and techniques to address shortcomings in any of the quality area discussed in the standard. So without further ado, let's start surfing.



SWEBOK

<http://www.swebok.org>

The Software Engineering Body of Knowledge was produced as a joint project under the ACM and IEEE. The

SWEBOK contains ten Knowledge Areas (KA) such as requirements, design, testing, construction, configuration management, etc. and breaks down each KA into topic areas. Each topic area contains a brief description of key concepts and provides bibliographic references to classic papers and publications that are considered the definitive works on each concept. As such, the SWEBOK captures what is called the “generally accepted knowledge” associated with software engineering. Massive in scale, work on the SWEBOK was started in 1998 and completed in 2004. Along the way, the SWEBOK was peer reviewed by over 500 software engineering practitioners in 42 countries who submitted approximately 9,000 comments against the document. All of the comments and their resolution are contained in a database at the SWEBOK site.

A work as important as the SWEBOK is certain to draw controversy and it certainly has. The SWEBOK has been criticized as a misguided attempt to set forth the mandatory knowledge required of all software engineers and that the SWEBOK could be used as a standard for licensing programmers. In 2000, the ACM stated its opposition to the licensing of programmers and withdrew from the joint committee working on the SWEBOK. Critics claim that unlike licensed civil engineers, the professional practice of software engineering is not as mature as other engineering disciplines and that licensing using the SWEBOK as the basis for that licensing would provide a false assurance of software engineering competence.

Nevertheless, the SWEBOK remains an invaluable source of information for any software engineer. If you are seeking the definitive source on a software engineering topic,

chances are you'll find the answer in the SWEBOK.

Software Engineering 2004 (SE2004)

<http://sites.computer.org/ccse>

Also known as the Computing Curriculum Software Engineering (CCSE), the SE2004 is similar to the SWEBOK in that it contains a bibliography of information that every software engineer should know. As opposed to the SWEBOK, the SE2004 is not a licensing standard, but a basic undergraduate curriculum in software engineering. Instead of the end-point for licensing, the SE2004 is a starting point for further (post-graduate) study.

SE2004 describes the Software Engineering Education Knowledge (SEEK), the body of knowledge that is the recommended course of study for an undergraduate program in software engineering. The SEEK draws on the knowledge areas of the SWEBOK and is intended to emphasize the academic underpinnings of the knowledge contained in the SWEBOK. In addition to the knowledge areas, SE2004 also contains recommended course sequences, guidelines for curriculum development, and recommendations for alternative teaching environments.

Several years ago I taught evening classes in programming, software engineering and systems engineering at a local college. I had to write my own syllabus. I wish the SEEK was available back then. Using this information, the smallest junior college could teach a world class course in just about any computer science subject.

Software Engineering 2004 Mozilla Firefox
http://sites.computer.org/ccse/

Software Engineering 2004
Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering
A Volume of the Computing Curricula Series
SE2004

Table of Contents

- [Release of the SE2004 Volume](#)
- [Reviewer Comments](#)
- [Overview](#)
- [SE2004 Steering Committee Members](#)
- [SE2004 History](#)

Release of SE2004 Volume

The SE2004 Volume is complete and ready for download: [Software Engineering 2004 Volume](#). We wish to extend a special **thank you** to all the volunteers who have participated in this effort! Appendix B of the SE2004 Volume has a list of volunteers.

Note: The earlier project name CCSE (Computing Curriculum Software Engineering) was replaced with the name SE2004 (Software Engineering 2004).

Reviewer Comments

Three general public reviews of the SE2004 Volume were carried out. Reviewer comments and comment responses from the SE2004 Steering Committee can be viewed or downloaded.

- [First General Review Comments](#)
- [Second & Third General Review Comments](#)

Overview

Computing Curriculum Project

In the Fall of 1998, the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) established the Joint Task Force on Computing Curricula 2001 (CC2001) to undertake a major review of curriculum guidelines for undergraduate programs in computing. The charter of the task force was expressed as follows:

"To review the Joint ACM and IEEE/CS Computing Curricula 1991 and develop a revised and enhanced version for the year 2001 that will match the latest developments of computing technologies in the past decade and endure through the next decade."

Certified Software Development Professional

http://www.computer.org/portal/site/ieeecs/menuitem.c5efb9b8ade9096b8a9ca0108bcd45f3/index.jsp?&pName=ieeecs_level1&path=ieeecs/education/certification&file=index.xml&xsl=generic.xsl&

The IEEE offers a Certified Software Development Professional (CSDP) certification at the IEEE web site. The URL for the CSDP page is four lines long, so just navigate to the IEEE site and select the certification sub-menu item from the “Career Development and Education” main menu item on the left. The CSDP certification requires a baccalaureate or equivalent degree and at least 9,000 hours of software engineering experience in six of eleven knowledge areas. The knowledge areas for the CSDP are almost the same as those in the SWEBOK with the addition of an area on professionalism and engineering economics.

It is important to draw a distinction between certification and licensing. Licensing implies you must pass an examination before you are allowed to practice the discipline. Certification allows one to work, but serves as proof of professionalism and carries

with it a code of professional behavior and ethics.

Software Engineering Institute

<http://www.sei.cmu.edu/managing/managing.html>

Not surprisingly, the Software Engineering Institute (SEI) is frequently mentioned in this column. After all, this is *Software Engineering Notes*, so it’s no wonder I often cite the SEI. The URL listed above for the management page at the SEI contains links to several resources that will help you improve software quality. The links are grouped into sections for quality; cost and schedule; tools; acquisition practices; and research. The SEI Capability Maturity Model Integration (CMMI) is listed under the quality section, but don’t go looking for canned processes under the CMMI. The CMMI is a handy

document to use if you are starting a new development or evaluating your current development procedures; it lists all the process areas your project should address. However, the CMMI does not tell you how to execute these process areas. For example, the CMMI says you **ought** to do configuration management, but it doesn’t tell you **how** to do configuration management. Check out the other sections or perhaps the SWEBOK for thoughts on implementing CMMI processes.

Delivering FORCEnet to the warfighter. Best practices - the road to higher performance and quality.

SSC San Diego Process Asset Library
(A repository of Center level standard process assets)

"If you can't describe what you are doing as a process, you don't know what you're doing."
— W. Edwards Deming

"The quality of a software system is governed by the quality of the process used to develop it."
— Watts Humphrey

"When you can measure what you are speaking about, and express it in numbers, you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."
— Lord Kelvin

"It is always fastest to do the job right the first time."
— Watts Humphrey

This is an official U.S. Navy web site.
This US Government system is subject to monitoring.
Web Page Reviewed and Approved by:
Tom Lapuzza, SSC San Diego Public Affairs Office (approval on file)

Mailing Address:
Commanding Officer
Space and Naval Warfare Systems Center
5360 Hull Street
San Diego, CA 92152-6001

Sub-web of SSC San Diego Public Web Site

The SSC San Diego Process Asset Library (PAL) is a sub-web of the Space and Naval Warfare Systems Center San Diego public Web site, which can be accessed by clicking the SSC San Diego link on the right. To return here, click About SSC San Diego and then Systems Engineering Process Office (SEPO).

Purpose

a library of military Data Item Descriptions (DIDs). A DID is a very handy document. It is basically a format for a software development artifact that includes and annotated outline for a given development document. For example, if you want to know what a software design document should look like, download DID DI-IPSC-81435A, entitled "Software Design Description". The DID contains a paragraph by paragraph description of the necessary content for requirements documents, test plans, version descriptions, operator's manuals, and many others. The SSC site is the next best thing to an IEEE standards library. And it's free.

No Silver Bullet

<http://www-inst.eecs.berkeley.edu/~maratb/readings/NoSilverBullet.html>

I'm not posting a screen shot of this link because it's just Fred Brooks' classic monolog, based on his years at IBM, that no magic tool or methodology will save us from bad software. This UC Berkeley site has reprinted this classic paper as part of a 2002 class in software development. In my opinion, it's a must read for every programmer.

The SEI pages have other resources to help you improve your software quality. The Software Engineering Information Repository (SEIR) (free registration required) contains all sorts of information on acquisition, development processes, measurements and metrics and more. The SEIR content is contributed by its members, so you'll find a lot of practical advice as well as the results of academic research. The SEIR may not tell you how to set up that configuration management system, but you'll find a lot of lessons learned and advice for tweaking your own CM process.

SPAWAR Systems Center San Diego

<http://sepo.spawar.navy.mil/>

When's the last time the US Department of Defense has done something nice for you? Well, if you are a software process engineer, here's something good. The Space and Naval Warfare (SPAWAR) Systems Center (SSC for short) has compiled a set of software process information in their Systems Engineering Process Office. SPAWAR has managed many of the Navy's large systems developments in communications, combat systems and other complex, mission critical systems. Okay, these aren't your typical banking, insurance; inventory or payroll systems, but the quality engineering techniques used to build these highly reliable, safety-critical systems can greatly improve the quality in your business system.

There's a wealth of information on the SSC site. You'll find everything from a CMMI tutorial to coding standards to project management plans to

Tom Van Vleck's Software Engineering Stories

<http://www.multicians.org/thvv/tvvswe.html>

Every once in a while I find an odd, out-of-the-way page with content so clever, I have to mention it. Tom Van Vleck's page of software engineering stories is one such page. I started to read a couple of the stories and the next

Software Engineering Stories

Here are some stories about software engineering. These stories are here for all to read and enjoy. Please send mail to me if you use them in classes or circulate them through your company. I'd also like a note if you link to these pages, so that I can let you know if they are moved, renamed, or improved.

[Three Questions About Each Bug You Find](#)
Published in ACM SIGSOFT Software Engineering Notes, vol 14 no 5 July 1999 pages 62-63. (07/99, 1sk, 1 comic)

[Cleaning up the Basement in the Dark](#)
Published in ACM Software Engineering Notes, April 1992. (04/92, 1k, 1 comic)

[It Can Be Done](#)
Published in IEEE Computer, May 1994. (05/94, 1k, 1 picture)

[The Evolution of My Thinking](#)
Facts in software engineering. (06/95, 4k)

[Nasty Question](#)
What do you do to make your product high quality? The four steps to quality. (06/95, 1k)

[A Few Minutes Discussion](#)
The conversation we should have had. (11/11/94, 8k)

[Notebooks](#)
What really happened? (07/95, 4k)

[NASA's \\$10,000 Drop of Blood](#)
A different Hubble story. (07/95, 5k)

[The Risks of Electronic Communication](#)
Advice to new electronic mail and netnews users. (08/95, 1.5k)

[Painting the Wall](#)
A person I didn't hire. (08/95, 5k)

[When the River is Polluted](#)
Find the root cause. (08/95, 5k)

[Structure Marking](#)
How the Multics directory structure was defended. (09/02/06, 19k, 1 figure)

[Perfection Pills](#)
A pharmacological approach to quality. (09/95, 5k, 1 comic)

[Counterpoint](#)
A lesson from John Cage. (11/11/94, 5k)

[Check Marks](#)
Observed high quality and how to do it. (08/95, 4k)

[Software Engineering Proverbs](#)
Collected proverbs. (09/95/06, 1sk, 1 cartoon)

[Software Engineering Reading List](#)
Some favorite books and papers on programming and quality. (03/04/06, 1sk)

[The Multics System Programming Process](#)
An invited paper (04/95, 2sk, 1 picture) for COMPCON '79 about our team's development process. Reprinted in the IEEE Tutorial on Software Maintenance.

[Software Engineering Comics](#)
Seven 6x9 comic strips for programmers. (07/94/95, 5k, 7 pictures)

[Hallelujah](#)
A technique for testing server programs. (02/11/06, 8k)

[Gause's Algorithm](#)
A date-related bug. (03/02/06, 0k, 1 picture)

[Management Tactics](#)

thing I knew it was an hour later and I still wasn't at the bottom of the page. The stories are amazing, inspirational, educational, sometimes funny, sometimes poignant, but all are a good read. They cover a lifetime of software engineering from the first story, an article from this very publication in 1989, to a thought-provoking piece on the nature of bugs posted just a few months ago.

When you're done with the stories, check out some of the



other content on Van Vleck's site. If you are an old guy like me, it will bring back some early memories of what programming was like before the dawn of OO anything. Van

Vleck's stories about his work on the IBM 7094 in the mid-60's remind us of just how far we've come. The \$3.5 million IBM 7094 shown here boasted 32K of 36-bit words of memory composed of a 3-bit prefix, 15-bit decrement, 3-bit tag, and 15-bit address. For those of you who are concerned with SOAP interfaces in a Service Oriented Architecture, this site will give you a whole 'nuther view of software engineering when interfaces were designed in bit positions in a computer word instead of elements in an XML document. Workmanship was not measured in the degree of inheritance of parent classes, but by the speed and efficiency of the memory manager you wrote for your application. After reading a few of Van Vleck's stories, you'll see just how far we've come.

Bad Software

<http://www.badssoftware.com>

The bad software site is the companion web site for the eponymous 1998 book by Cem Kaner and David Pels. The site is somewhat dated, but contains articles on the legal aspects of bad software. The book itself was written to inform consumers of their rights in the event that purchased software is found to be defective. Several sections of the book are reprinted on the web site and provide advice for the owners of bad software. However, the software practitioner will be most interested in site section on the "Law of Software Quality" and the collection of court cases involving bad software. Dr. Kaner is both a software engineer (and professor of computer science at the Florida

Institute of Technology) and a lawyer, so he knows how to build bad software **and** how to sue the builders of bad software.

The site and the book were originally intended to discuss the impact of the Uniform Computer Information Transactions Act (UCITA) of 1999. The act was widely opposed by consumer advocates who claimed that a consumer, by clicking on the End User License Agreement (EULA) would lose the rights to sue the software vendor if the user was damaged by the software. Eventually the law was only adopted in Maryland and Virginia and several states passed laws specifically barring the provisions of the original UCITA proposal. The furor has mostly died down over the past few years, but this site contains a nice summary of UCITA activity.

Salon Software

<http://dir.salon.com/topics/software/>

Moving from the academic to the practical, the software page on the Salon website contains several great articles on programming and software engineering. The most recent article by Andrew Leonard is an interview with Salon co-founder Scott Rosenberg about his book, "Dreaming in Code", the story of three years of development on the Chandler project. Entitled "Software is Hard", the Salon article ties together themes from Brooks' "The Mythical Man Month" and Yourdon's "Death March" to once again tell the story of how inattention to the basics of software engineering can result in an undisciplined, out of control project.

Bad Software: A Consumer Protection Guide

[Main | List of Articles | Bad Software | UCITA | Law of Software Quality | Digital Signatures | Bookstore | Court Cases | Links | Press Releases | About Us | What's New]

David and I (Cem) sometimes have computer problems too. I'm currently (07/2005) having my worst purchase/support experience in 24 years. You might be interested to see how I use a blog and various consumer advocates to fight back. For details, see my [ALIENWARE STUCKS](#) blog.

Bad Software

Bad Software is a book about getting your money's worth when you buy computer software. We wrote it to help you get a refund, support, or compensation for significant losses caused by defective software. Bad Software will help you explore your legal rights. We spend more time suggesting ways to negotiate with publishers than sue them, but if they won't play fair, you can make them wish that they had.

THIS WEB SITE was intended as a gathering point for information about software consumer protection. It morphed into one of the main sites for opposition to the [Uniform Computer Information Transactions Act \(UCITA\)](#). UCITA was approved by the National Conference of Commissioners on Uniform State Laws in 1999, but approved in only two states and then only with amendments. The main opposition site today is the [AFFECT Coalition](#).

To our surprise, we still get thousands of hits per month at this site, downloading historical documents. We intend to leave those links intact for a long time. Eventually, this page will be reworked, probably as a course site for my computer law/ethics courses, that focuses on issues in software consumer protection.

What follows is the page as archived in April, 2001.

[This site gathers information about software consumer protection, your rights, and legislation that will affect you.] For example:

- **Chapter 1 of our book: You have rights when you buy defective software.** Currently, Article 2 of the Uniform Commercial Code governs sales of packaged software. You have rights under Article 2. For example, if you buy a program, take it home, and discover obvious defects in your first day of use, you can take the program back for a refund. Some stores will tell you that the Copyright Act prevents this. That just ain't so. For more information, and tips on what you can do, read [Chapter 1 of our book](#).
- **The Uniform Computer Information Transactions Act** is a new law that is intended to govern all contracts involving computer software and information that you obtain electronically (website, CD, etc.) It also stretches early to cover computers, printers and other computer peripherals, and software that is embedded in goods (such as fuel injection software in your car.) This proposed law grants new intellectual property rights to software and information publishers (going well beyond the powers they have under the United States Copyright and Patent laws) and makes it almost impossible to hold vendors of defective products accountable for defects and misrepresentations. (Note: Much of the material at this site is archival, presenting documents up to July, 1999. UCITA passed NCCUSL in July, 1999 and went to the state legislatures. [The AFFECT Coalition](#) is coordinating the opposition to NCCUSL in the state legislatures.)
- Our new papers:
 - [The Problem of Embedded Software \(Part 1\)](#) January 2001 with Professor Phil Koopman, published in UCC Bulletin (February, March issues)
 - [The Problem of Embedded Software \(Part 2\)](#) March 2001 with Professor Phil Koopman, published in UCC Bulletin (April issue)
 - [Proposed Article 2 Revision Fall Short for Embedded Software](#), November 2000, with Professor Phil Koopman (for UCC 2 drafting committee)
 - [Comments to the Federal Trade Commission](#) submitted for its hearing on software warranties and consumer protection
 - [Software Engineering and UCITA](#), published in the Journal of Computer Law
 - [Why You Should Oppose UCITA](#), published in the Computer Lawyer

[Main | List of Articles | Bad Software | UCITA | Law of Software Quality | Digital Signatures | Bookstore | Court Cases | Links | Press Releases | About Us | What's New]

The screenshot shows the Salon website in a Mozilla Firefox browser window. The page features a search bar, navigation links, and a main article titled "Software is hard" by Scott Rosenberg. The article discusses the challenges of programming and includes several sub-articles with small icons: "Indian programmer woes", "The Shlemiel way of software", "When machines breed", and "We don't support that".

how to use those techniques, he takes a higher level view of how the software you write will impact the user. The result is an almost Mark Twain folksy approach to software engineering. It's a refreshing change from the rather dry SEI material.

Usability Net

<http://www.usabilitynet.org/home.htm>

Following up on Green's assertion that the real hallmark of quality software is the degree of user satisfaction, the Usability Net project provides recommendations and advice on improving end user usability via user centered design. Supported by the European Union, Usability Net has collected an impressive array of resources for improving software usability, particularly web application usability.

There are a number of cool features on this site, but my favorite is the interactive methods table. The user-centric methods supporting software development are displayed in the table under the software development phase where the technique is most valuable. For example, Usability Net recommends

The Salon site has other, easy-to-read articles that provide insight to the practical, non-academic side to the software engineering problem.

interactive methods table. The user-centric methods supporting software development are displayed in the table under the software development phase where the technique is most valuable. For example, Usability Net recommends

Robelle Solutions

<http://www.robelle.com/library/papers>

Bob Green of Robelle Solutions has posted some common-sense monographs on software quality on his company's web site. The papers "Building Better Software" and "Improving Software Quality" are a couple of well-written editorials on software quality and how to achieve it. I present these works as another counterpoint to the academic and theoretical references presented earlier. Green's firm specializes in software for the HP 3000, a niche market if I've ever heard one, yet the lessons he has learned over the years and the way he explains how quality is manifested in his products are brought to life in a way any programmer can understand.

Green cites the classic software engineering references (Brooks, Dijkstra, etc.) in his articles, but instead of a standard instruction on

The screenshot shows the Robelle Solutions website in a Mozilla Firefox browser window. The page features a navigation bar with the Robelle logo and links for "Data Extraction" and "Easy Text Editing". The main content area is titled "Improving Software Quality" and includes a graphic of a staircase labeled "Steps To Software Quality". Below the graphic, the author is identified as Robert Green, and the company is Robelle Solutions Technology Inc. The text discusses the author's experience with software quality and provides an outline of the paper's content.



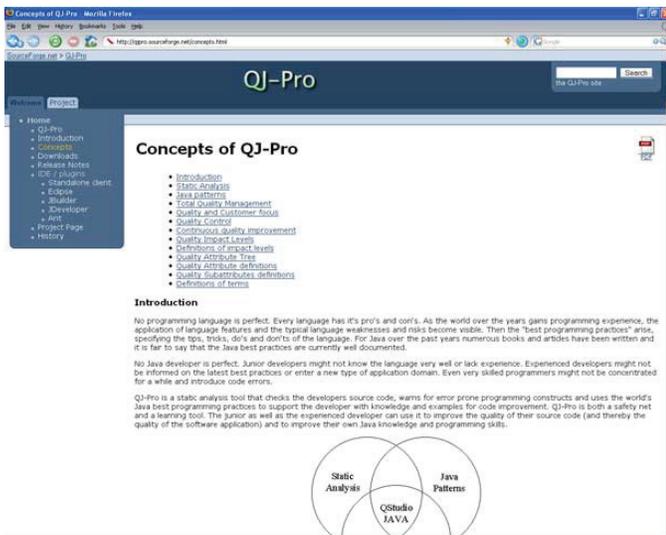
User Surveys for Requirements Analysis and Storyboarding for the Design phase. All of the methods are hyperlinked to pages providing details on how to employ the methods in your development. But wait, there's more! The page allows you to select from three conditions, "limited time/resources", "no direct access to users", and/or "limited skills/expertise" and the table will highlight the techniques that are most appropriate under your selected circumstances.

They practice what they preach at the Usability Net site. Clicking on the WAMMI (no, I don't know what WAMMI stands for) button, found at various locations on the site, takes you to a web usability user survey that Usability Net uses to improve the quality of the site.

QJ-Pro

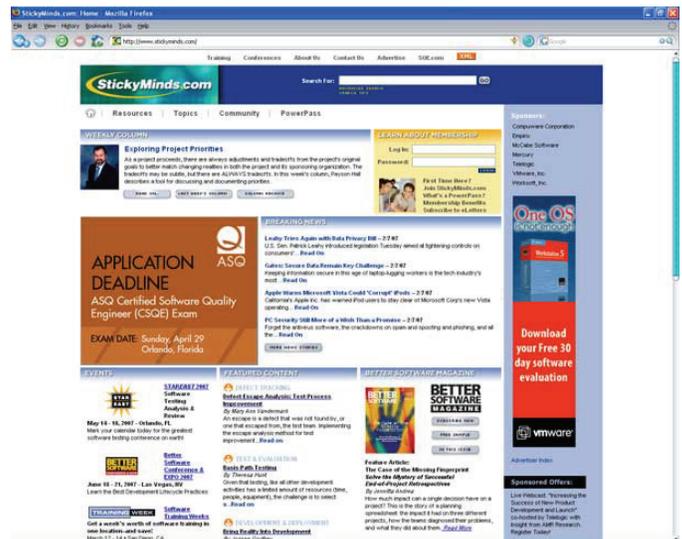
<http://qjpro.sourceforge.net>

I've covered many software quality tools in the past, including coding standards checkers and static analysis tools, but for some reason, I've missed QJ-Pro. This Java code analyzer is designed to check quality attributes such as reliability, maintainability, testability, and portability. It can



be used to measure adherence to your project's coding standards and comes with a default set of coding standards based on good programming practice. QJ-Pro can be run as a standalone client or as a plug-in to Eclipse, JBuilder, JDeveloper, or Ant. QJ-Pro is the static analysis engine used in several commercial source code analysis products.

At the QJ-Pro website, you can browse the CVS repository containing the full source code for QJ-Pro and you can also download binary executables for Windows, Linux, and Solaris. In addition to the QJ-Pro tool itself, the QJ-Pro site has a neat one-page discussion of the concepts of static analysis and code quality assessment. Unlike this column, the QJ-Pro discussion of code quality is well-written, very concise and fairly comprehensive.

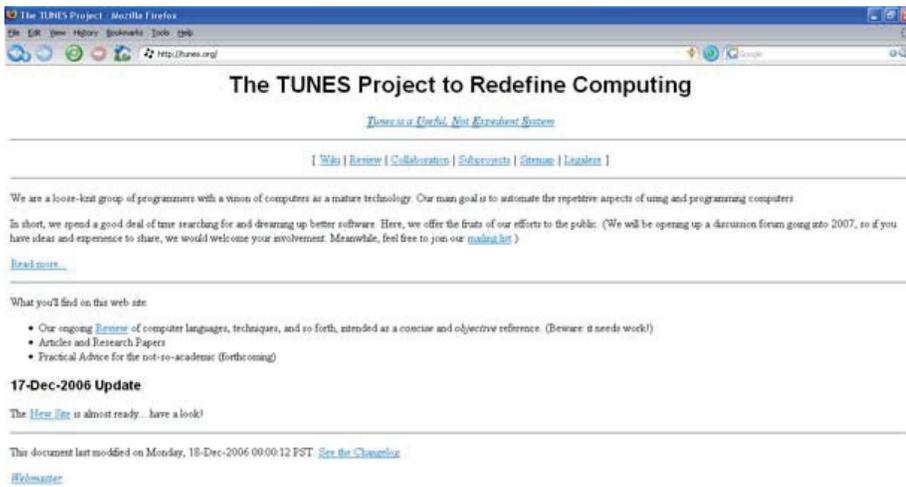


StickyMinds.com

<http://www.stickyminds.com>

I don't usually include commercial sites in this column, but the StickyMinds site is an exception. There is a lot of content here to assist with the software quality problem. From the rather pricy (\$75 per year) Better Software magazine to the free web site content, StickyMinds is a rich resource for the software professional who is looking to improve software quality. The homepage has the typical advertisements for training courses, consulting assistance, and events. And there are news articles of interest with the standard RSS feed. But the good stuff is under the menu items along the top of the web page.

The site is sponsored by software quality tools vendors such as Compuware, McCabe, Mercury, Telelogic and others. If you look under the resources tab, you'll find a tools guide that lists hundreds of quality tools, not just those offered by the site sponsors. I've bookmarked this site just for the tools guide. I can't tell you how many times I've wanted to find a tool that I've written about in a past article, but just couldn't remember the name of the tool. The StickyMinds site fixes that for me, by providing a sticky place where I



can scroll through the tools listings to jog my memory. Once I've found the tool again, the listing usually has a link that takes me right to the tool's web site.

The TUNES Project

<http://tunes.org/>

The TUNES project to redefine computing has been bumping along for many (by Internet standards) years. Quoting from the site:

“TUNES is a project to replace existing Operating Systems, Languages, and User Interfaces by a completely rethought Computing System, based on a fully reflective architecture with standard support for unification of system abstractions, security based on formal proofs from explicit negotiated axioms, higher-order functions, self-extensible syntax, fine-grained composition, distributed networking, orthogonally persistent storage, fault-tolerant computation, version-aware identification, decentralized (no-kernel) communication, dynamic code regeneration, high-level models of encapsulation, hardware-independent exchange of code, migratable actors, yet (eventually) a highly-performant set of dynamic compilation tools”.

In short, a whole new way of looking at computing.

It's an exciting concept. We've been working for decades with things like flowcharts, data flow diagrams, functionally decomposed algorithmic languages, objects, and service

oriented architectures. So far, none of those approaches have produced Brooks' Silver Bullet. So why not take a radical new look at the computing problem? TUNES is attempting to do just that. The TUNES site collects several subprojects that are working towards this redefinition and has a lot of interesting reading that will give you a new way of looking at computing.

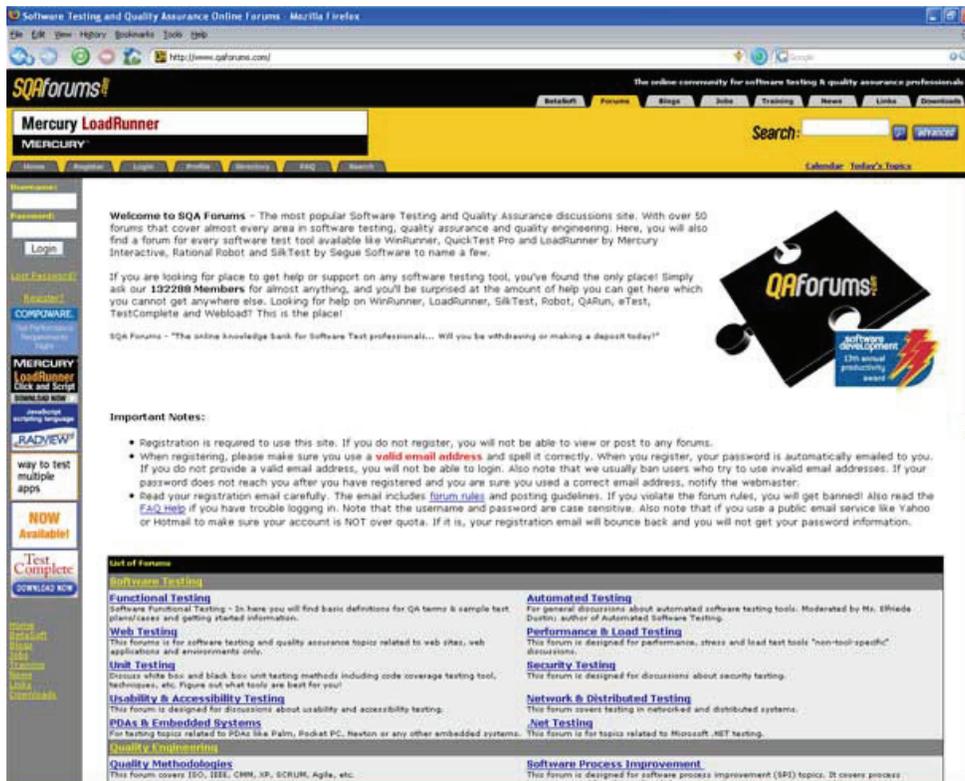
Epigram

<http://e-pig.org>

Epigram represents a new approach to programming languages that focuses on adding semantics into types. The Epigram project itself consists of a dependently typed programming language and an interactive programming environment. Quoting from Conor McBride's Epigram tutorial on the site:

“Abstraction and application, tupling and projection: these provide the `software engineering' super-structure for programs, and our familiar type systems ensure that these operations are used compatibly. However, sooner or later, most programs inspect data and make a choice -- at that point our familiar type systems fall silent. They simply can't talk about specific data. All this time, we thought our programming was strongly typed, when it was just our *software engineering*. In order to do better, we need a static language capable of expressing the significance of particular values in legitimizing some computations rather than others.”





establishing a software quality assurance or software process improvement program for any new software development.

The QAForums site contains a number of very active discussion groups. Many of the topics areas have thousands of posts and multiple threads. The posts are recent and the discussions appear lively without rancor. The screen shot only lists a few of the resources available. There are a lot more groups listed further down the page that are not visible in the illustration.

The American Society for Quality

<http://www.asq.org>

The American Society for Quality (ASQ) is the ACM for quality assurance professionals. Their web site features a wide variety of educational material on all aspects of quality, including software quality. In addition special sections of the site are dedicated to the application of quality assurance techniques in several industry sectors such as education, healthcare, government and others. Special sections of the site contain information on using the Six Sigma approach to quality and information on preparations for competition for the Malcolm Baldrige National Quality Award.

Epigram uses a type system that allows the programmer to express the behavior of the program in the types and uses the type checker to ensure the program is well-behaved.

It's an interesting approach and this short blurb does not do it justice. Will it work? Heck if I know, but Epigram certainly represents a different way of looking at the programming problem and is representative of some of the new work being done to attack the quality problem.

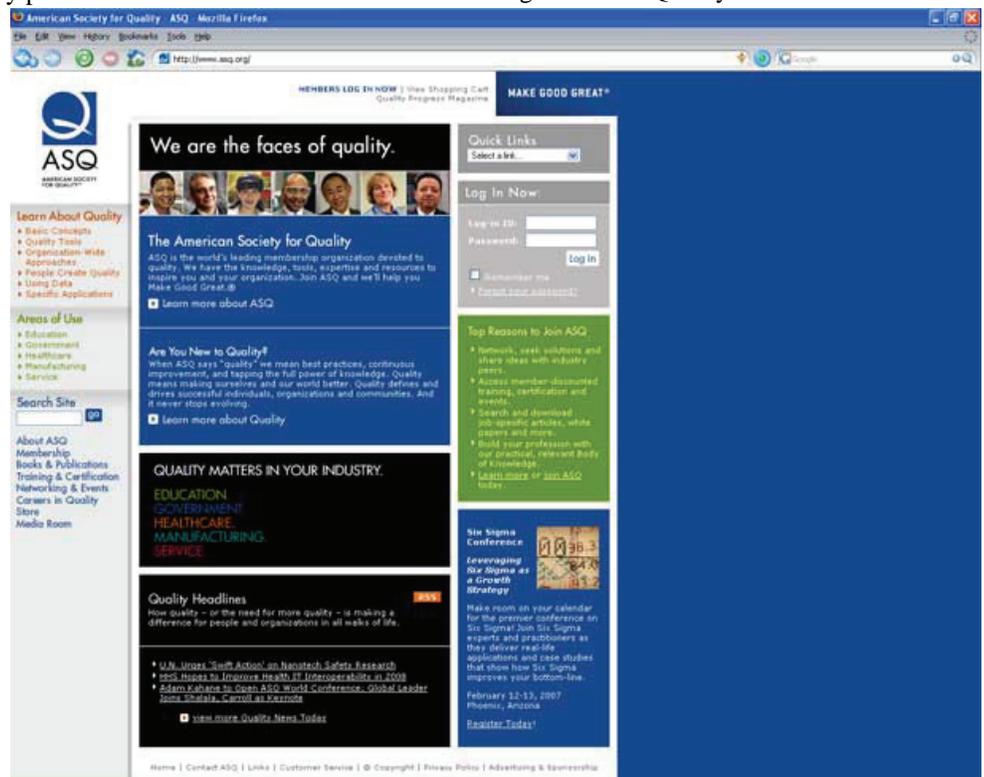
educational material on all aspects of quality, including software quality. In addition special sections of the site are dedicated to the application of quality assurance techniques in several industry sectors such as education, healthcare, government and others. Special sections of the site contain information on using the Six Sigma approach to quality and information on preparations for competition for the Malcolm Baldrige National Quality Award.

QAForums

<http://www.qaforums.com>

I always like to throw in a forum site where the topics and conversation can cover a lot of ground. The Quality Assurance Forum covers various topics of interest to the QA professional. The site features extensive software testing discussions including discussion groups talking about test tools. Registration is free and required to view any of the topic groups in the QA Forum.

This is an outstanding resource for software testers as well as quality assurance engineers. There are in depth discussions on software unit testing, testing methodologies, metrics collection and software process improvement. This would be a good site to visit when tasked with



offers two assessment packages that will evaluate your organization's maturity in software testing and software quality assurance. Membership is \$120 for an individual and \$1,000 for a corporate membership. Members receive a monthly newsletter, a quarterly journal and access to the Process Workshop, an online collection of process white papers.

INCOSE

<http://www.incose.org>

This rather sparse looking website sits atop a ton of content. The website for the International Council on Systems Engineering (INCOSE) provides a great deal of basic advice on the systems engineering process. Some of the content is restricted to members only, but there's still a lot of good stuff for non-members. While the StickyMinds site lists quality tools, the INCOSE site has an extensive list of

systems engineering tools. The tools matrix contains entries for almost 1500 systems engineering tools with a query capability to help you find what you are looking for. The INCOSE team has conducted separate tools surveys for requirements management tools, architecture tools, and measurement tools. The survey pages compare the features of the tools side by side, so you can determine which tool best fits your needs.

You can also reach the SEBOK, the Systems Engineering Body of Knowledge through the INCOSE pages. Actually the link (<http://g2sebok.incose.org/>) is to the Guide to the SEBOK (using the ugly acronym G2SEBOK). Unlike the SWEBOK, the SEBOK is a hierarchical listing of topics with a brief discussion of the body of knowledge about the topic. The SEBOK lacks the bibliographic depth of the SWEBOK.

End Note

Well, it looks like I'm out of time and space for this month's surfing trip. I hope some of the sites here will provide you with some new ideas on how to improve the quality of your software developments.

Quality Assurance Institute

<http://www.qaiworldwide.org>

The Quality Assurance Institute is the worldwide professional organization that sponsors conferences, training, and certification in quality assurance. The Institute