

**PREVENTING SOFTWARE FAILURE:
PROBLEMS NOTED IN BREACH OF CONTRACT LITIGATION**

Draft 6 – October 11, 2008

Abstract

From working as an expert witness in a number of lawsuits where large software projects were cancelled or did not operate correctly when deployed, four major problems occur repeatedly: 1) Accurate estimates are not produced or are overruled; 2) Requirements changes are not handled effectively; 3) Quality control is deficient; 4) Progress tracking fails to alert higher management to the seriousness of the issues. There are often other problems as well, but these four always occur in breach of contract litigation.

Depositions and testimony in every lawsuit revealed that many software engineers and some managers on troubled projects knew about the problems months before the projects were terminated or the failures were clearly evident. Depositions and testimony also showed that normal project status reports did not elevate the problems to higher management or to customers. In fact many progress reports gave false indications that the projects were on track. This article discusses the topics that should be included in software project tracking reports to minimize failures and major problems.

Capers Jones, Chief Scientist Emeritus
Software Productivity Research LLC.

Web www.SPR.com
Email CJonesIII@CS.com

**Copyright 2007-2008 © by Capers Jones.
All Rights Reserved.**

PREVENTING SOFTWARE FAILURE: PROBLEMS NOTED IN BREACH OF CONTRACT LITIGATION

Introduction

There are millions of software projects in the world, and thousands of software technologies available. This means that research into topics that affect software project outcomes is of necessity a complicated issue. By concentrating on the extreme ends of possible results, it is easier to see the root causes of success and failure. Projects that set records for productivity and quality are at one of the scale. Projects that are cancelled or have problems severe enough for litigation are at the other end of the scale. This article concentrates on “worst practices” or the factors that most often lead to failure and litigation.

For the purposes of this article, software “failures” are defined as software projects which met any of these attributes:

1. Termination of project due to cost or schedule overruns.
2. Schedule or cost overruns in excess of 50% of initial estimates
3. Applications which, upon deployment, fail to operate safely.
4. Law suits brought by clients for contractual non-compliance.

Although there are many factors associated with schedule delays and project cancellations, the failures that end up in court always seem to have four major deficiencies:

1. Accurate estimates were either not prepared or were rejected.
2. Change control was not handled effectively.
3. Quality control was inadequate.
4. Progress tracking did not reveal the true status of the project.

Let us consider each of these topics in turn.

Estimating Problems

Although cost estimating is difficult there are a number of commercial software cost estimating tools that do a capable job: COCOMO II, KnowledgePlan, Price-S, SEER, SLIM, SoftCost, are examples.

However just because an accurate estimate can be produced using a commercial estimating tool that does not mean that clients or executives will accept it. In fact from information presented during litigation, about half of the cases did not produce accurate estimates at all and did not use estimating tools. However, the other half had accurate

estimates, but they were rejected and replaced by forced estimates based on business needs rather than team abilities.

The main reason that accurate estimates were rejected and replaced was the absence of supporting historical data. Without history, even accurate estimates may not be convincing. A lack of solid historical data makes project managers, executives, and clients blind to the realities of software development.

Suppose you are a project manager responsible for a kind of software project which no company in the world has ever been able to build in less than 36 calendar months. As a responsible manager, you develop a careful estimate and critical path analysis, and tell the client and your own executives that you think the project will require 36 to 38 months for completion.

What will often occur is an arbitrary rejection of your plan, and a directive by either the client or by your own executives to “finish this project in 18 months.” The project in question will usually be a disaster, it will certainly run late, and from the day you receive the directive the project is essentially doomed.

A situation such as this was one of the contributing factors to the long delay in opening the new Denver Airport. Estimates for the length of time to complete and debug the very complex baggage handling software were not believed, according to the article on “Software’s Chronic Crisis” in the September 1994 issue of Scientific American Magazine by T. Wayt Gibbs.

For more than 60 years the software industry lacked a solid empirical foundation of measured results that was available to the public. Thus almost every major software project is subject to arbitrary and sometimes irrational schedule and cost constraints. However the International Software Benchmarking Standards Group (ISBSG), a non-profit organization, has started to improve this situation by offering schedule, effort, and cost benchmark reports to the general public. This data is available in both CD and paper form. Currently more than 4,000 projects are available, and new projects are added at a rate of perhaps 500 per year.

There are other collections of software benchmark data, such as those gathered by the Gartner Group, by the David’s Consulting Group, by Software Productivity Research, and by other companies too. However, this data is usually made available only on a subscription basis to specific clients of the organizations. The ISBSG data, by contrast, is available to the general public.

Changing Requirements

The average rate at which software requirements change is about 1% per calendar month. Thus for a project with a 12 month schedule, more than 10% of the final delivery will not have been defined during the requirements phase. For a 36 month project, almost a third of the features and functions may have come in as afterthoughts.

These are only average results. The author has observed a three-year project where the delivered product exceeded the functions in the initial requirements by about 289%. It is of some importance to the software industry that the rate at which requirements creep or grow can now be measured directly by means of the function point metric. This explains why function point metrics are now starting to become the basis of software contracts and outsource agreements.

(Data on requirements creep was noted in all of the breach-of-contract cases where the author participated as an expert witness. Rates of requirements creep are measured by many companies. Among the author's clients, more than 100 companies measure requirements creep using function point metrics.)

The current state of the art for dealing with changing requirements includes the following:

- Effective mapping of business needs to the proposed applications
- Estimating the number and rate of development changes before starting
- Using function point metrics to quantify changes
- A joint client/development change control board or designated domain experts
- Full time involvement by user representatives for Agile projects
- Use of joint application design (JAD) to minimize downstream changes
- Use of formal requirements inspections to minimize downstream changes
- Use of formal prototypes to minimize downstream changes
- Planned usage of iterative development to accommodate changes
- Formal review of all change requests
- Revised cost and schedule estimates for all changes > 10 function points
- Prioritization of change requests in terms of business impact
- Formal assignment of change requests to specific releases
- Use of automated change control tools with cross-reference capabilities

Unfortunately in projects where litigation occurred, requirements changes were numerous but their effects were not properly integrated into cost, schedule, and quality estimates. As a result, unplanned slippages and overruns occurred.

In several cases, the requirements changes had not been formally included in the contracts for development, and the clients refused to pay for changes that substantially affected the scope of the projects. One case involved 82 changes that totaled to more than 2,000 function points or about 20% of the original size of the initial requirements. Although the contract did include clauses for funding "out of scope" changes, the defendant asserted that the 82 changes were merely refinements rather than changes. It is obvious that contracts need to be very specific about what constitutes "change."

Since the defect potentials for changing requirements are larger than for the original requirements by about 10%, and since defect removal efficiency for changing requirements is lower by about 5%, projects with large volumes of changing

requirements also have severe quality problems, which are usually invisible until testing begins. When testing begins, the project is in serious trouble because it is too late to bring the schedule and cost overruns under control.

One of the observed byproducts of the usage of formal joint application design JAD sessions is a reduction in downstream requirements changes. Rather than having unplanned requirements surface at a rate of 1% to 3% per month, studies of JAD by IBM and other companies have indicated that unplanned requirements changes often drop below 0.5% per month due to the effectiveness of the JAD technique.

Prototypes are also helpful in reducing the rates of downstream requirements changes. Normally key screens, inputs, and outputs are prototyped so users have some “hands on” experience with what the completed application will look like.

The Agile method of having a full-time user representative attached to the project is also valuable, if this is possible. However for very large projects with perhaps millions of potential users (such as Microsoft Vista) one user cannot speak for every user. Therefore truly representative customer involvement is feasible only for projects with a fairly small number of users who are likely to utilize the application in similar ways.

Requirements changes will always occur for large systems. It is not possible to freeze the requirements of any real-world application and it is naïve to think this can occur. Therefore leading companies are ready and able to deal with changes, and do not let them become impediments to progress. For projects developed under contract, the contract itself must include unambiguous language for dealing with changes.

Quality Problems

It is dismaying to observe the fact that one of the most effective technologies in all of software is never used on projects that turn out to be disasters and end up in court. Formal design and code inspections have a 50 year history of successful deployment on large and complex software systems. All “best in class” software producers utilize software inspections. The measured defect removal efficiency of inspections is more than twice that of most forms of software testing (i.e. about 65% for inspections versus 30% for most kinds of testing).

(The term “best in class” is subjective. In this article and other studies by the author, it refers to projects that are in the top 15% of all projects measured in terms of quality, schedules, and productivity rates at the same time.)

Effective software quality control is the most important single factor that separates successful projects from delays and disasters. The reason for this is because finding and fixing bugs is the most expensive cost element for large systems, and takes more time than any other activity.

Successful quality control involves defect prevention, defect removal, and defect measurement activities. The phrase “*defect prevention*” includes all activities that

minimize the probability of creating an error or defect in the first place. Examples of defect prevention activities include the Six-Sigma approach, joint application design (JAD) for gathering requirements, usage formal design methods, usage of structured coding techniques, and usage of libraries of proven reusable material.

The phrase “*defect removal*” includes all activities that can find errors or defects in any kind of deliverable. Examples of defect removal activities include requirements inspections, design inspections, document inspections, code inspections, automated static analysis of code, complexity analysis, and all kinds of testing.

Some methods can operate in both defect prevention and defect removals domains simultaneously. The most notable example of a method that is effective in both defect prevention and defect removal roles is that of formal design and code inspections. Inspections are the top-ranked defect removal method in terms of efficiency. Also, participation in formal inspections is one of the top methods for defect prevention. After participation in several design and code inspections, participants spontaneously avoid the kinds of problems that were encountered. The net effect of inspections in terms of defect prevention is a reduction of about 50% of potential defects.

Both “defect potentials” and “defect removal efficiency” should be measured for every project. The “defect potentials” are the sum of all classes of defects; i.e. defects found in requirements, design, source code, user documents, and “bad fixes” or secondary defects. It would be desirable to include defects in test cases too, since there may be more defects in test libraries than in the applications being tested.

The phrase “defect removal efficiency” refers to the percentage of defects found before delivery of the software to its actual clients or users. If the development team finds 900 defects and the users find 100 defects in a standard time period after release (normally 90 days) then it is obvious that the defect removal efficiency is 90%.

The author strongly recommends that defect removal efficiency levels be included in software outsource and development contracts, with 95% being a proposed acceptable level of defect removal efficiency. (The U.S. average is only about 85% so 95% is a significant improvement. For some mission-critical applications, a higher level such as 99% might be used, but it is technically challenging to achieve removal efficiency levels much higher than about 97%.) Following are examples of defect prevention and defect removal:

Defect Prevention

- Joint application design (JAD) for gathering requirements
- Thorough analysis of business and technical needs
- Formal architectural analysis before starting design
- Formal design methods
- Structured coding methods
- Formal defect and quality estimation

- Formal test plans
- Formal test case construction
- Participation in formal inspections
- Formal change management methods
- Security analysis for the application
- Six-Sigma approaches (customized for software)
- Utilization of the Software Engineering Institute's capability maturity model (CMM) or (CMMI).
- Utilization of the new team and personal software processes (TSP, PSP)
- Utilization of Quality Function Deployment (QFD)

Defect Removal

- Requirements inspections
- Design inspections
- Document inspections
- Code inspections
- Test plan and test case inspection
- Automated static analysis
- Defect repair inspection
- Software quality assurance reviews
- Unit testing (manual and automated)
- Component testing
- New function testing
- Regression testing (manual and automated)
- Performance testing
- Security testing
- System testing
- Acceptance testing

The combination of defect prevention and defect removal activities leads to some very significant differences in the overall numbers of software defects compared between successful and unsuccessful projects. For projects in the 10,000 function point range, the successful ones accumulate development totals of around 4.0 defects per function point and remove about 95% of them before delivery to customers. In other words, the number of delivered defects is about 0.2 defects per function point or 2,000 total latent defects. Of these about 10% or 200 would be fairly serious defects. The rest would be minor or cosmetic defects.

By contrast, the unsuccessful projects accumulate development totals of around 7.0 defects per function point and remove only about 80% of them before delivery. The number of delivered defects is about 1.4 defects per function point or 14,000 total latent defects. Of these about 20% or 2,800 would be fairly serious defects. This large number of latent defects after delivery is very troubling for users. The large number of delivered defects is also a frequent cause of litigation.

Unsuccessful projects typically omit design and code inspections and depend purely on testing. The omission of up-front inspections causes three serious problems: 1) The large number of defects still present when testing begins slows down the project to a standstill; 2) The “bad fix” injection rate for projects without inspections is alarmingly high; 3) The overall defect removal efficiency associated with only testing is not sufficient to achieve defect removal rates higher than about 80%.

Software Milestone Tracking

Readers of this article who work for the Department of Defense or for a defense contractor will note that the “earned value” approach is only cited in passing. There are several reasons for this. First, none of the lawsuits where the author was an expert witness involved defense projects so the earned-value method was not utilized. Second, although the earned-value method is common in the defense community, its usage among civilian projects including outsourced projects is very rare. Third, empirical data on the effectiveness of the earned-value approach is sparse. A number of defense projects that used earned-value methods have run late and been over budget. There are features of the earned-value method that would seem to improve both project estimating and project tracking, but empirical results are sparse.

Once a software project is underway, there are no fixed and reliable guidelines for judging its rate of progress. The civilian software industry has long utilized ad hoc milestones such as completion of design or completion of coding. However, these milestones are notoriously unreliable.

Tracking software projects requires dealing with two separate issues: 1) Achieving specific and tangible milestones; 2) Expending resources and funds within specific budgeted amounts.

Because software milestones and costs are affected by requirements changes and “scope creep” it is important to measure the increase in size of requirements changes, when they affect function point totals. However there are also requirements changes that do not affect function point totals, which are termed “requirements churn.” Both creep and churn occur at random intervals. Churn is harder to measure than creep and is often measured via “backfiring” or mathematical conversion between source code statements and function point metrics.

As of 2008 there are automated tools available that can assist project managers in recording the kinds of vital information needed for milestone reports. These tools can record schedules, resources, size changes, and also issues or problems.

For an industry now more than 50 years of age, it is somewhat surprising that there is no general or universal set of project milestones for indicating tangible progress. From SPR’s assessment and baseline studies, following are some representative milestones that have shown practical value.

Note that these milestones assume an explicit and formal review connected with the construction of every major software deliverable. Formal reviews and inspections have the highest defect removal efficiency levels of any known kind of quality control activity, and are characteristics of “best in class” organizations.

Table 1: Representative Tracking Milestones for Large Software Projects

1. Requirements document completed
2. Requirements document review completed
3. Initial cost estimate completed
4. Initial cost estimate review completed
5. Development plan completed
6. Development plan review completed
7. Cost tracking system initialized
8. Defect tracking system initialized
9. Prototype completed
10. Prototype review completed
11. Complexity analysis of base system (for enhancement projects)
12. Code restructuring of base system (for enhancement projects)
13. Functional specification completed
14. Functional specification review completed
15. Data specification completed
16. Data specification review completed
17. Logic specification completed
18. Logic specification review completed
19. Quality control plan completed
20. Quality control plan review completed
21. Change control plan completed
22. Change control plan review completed
23. Security plan completed
24. Security plan review completed
25. User information plan completed
26. User information plan review completed
27. Code for specific modules completed
28. Code inspection for specific modules completed
29. Code for specific modules unit tested
30. Test plan completed
31. Test plan review completed
32. Test cases for specific test stage completed
33. Test case inspection for specific test stage completed
34. Test stage completed
35. Test stage review completed
36. Integration for specific build completed
37. Integration review for specific build completed
38. User information completed
39. User information review completed

40. Quality assurance sign off completed
41. Delivery to beta test clients completed
42. Delivery to clients completed

The most important aspect of table 1 is that every milestone is based on completing a review, inspection, or test. Just finishing up a document or writing code should not be considered a milestone unless the deliverables have been reviewed, inspected, or tested.

In the litigation where the author worked as an expert witness, these criteria were not met. Milestones were very informal and consisted primarily of calendar dates, without any validation of the materials themselves.

Also, the format and structure of the milestone reports were inadequate. At the top of every milestone report problems and issues or “red flag” items should be highlighted and discussed first.

During depositions and review of court documents, it was noted that software engineering personnel and many managers were aware of the problems that later triggered the delays, cost overruns, quality problems, and litigation. At the lowest levels, these problems were often included in weekly status reports or discussed at team meetings. But for the higher-level milestone and tracking reports that reached clients and executives, the hazardous issues were either omitted or glossed over.

A suggested format for monthly progress tracking reports delivered to clients and higher management would include these sections:

Suggested Format for Monthly Status Reports for Software Projects

1. Status of last months “red flag” problems
2. New “red flag” problems noted this month

3. Change requests processed this month versus change requests predicted
4. Change requests predicted for next month
5. Size in function points for this months change requests
6. Size in function points predicted for next month’s change requests

7. Schedule impacts of this month’s change requests
8. Cost impacts of this month’s change requests
9. Quality impacts of this month’s change requests

10. Defects found this month versus defects predicted
11. Defects predicted for next month

12. Costs expended this month versus costs predicted
13. Costs predicted for next month

14. Deliverables completed this month versus deliverables predicted
15. Deliverables predicted for next month

Although the suggested format somewhat resembles the items calculated using the earned value method, this format deals explicitly with the impact of change requests and also uses function point metrics for expressing costs and quality data.

An interesting question is the frequency with which milestone progress should be reported. The most common reporting frequency is monthly, although exception reports can be filed at any time that it is suspected that something has occurred that can cause perturbations. For example, serious illness of key project personnel or resignation of key personnel might very well affect project milestone completions and this kind of situation cannot be anticipated.

It might be thought that monthly reports are too far apart for small projects that only last six months or less in total. For small projects weekly reports might be preferred. However, small projects usually do not get into serious trouble with cost and schedule overruns, whereas large projects almost always get in trouble with cost and schedule overruns. This article concentrates on the issues associated with large projects. In the litigation where the author has been an expert witness, every project under litigation except one was larger than 10,000 function points in size.

The simultaneous deployment of software sizing tools, estimating tools, planning tools, and methodology management tools can provide fairly unambiguous points in the development cycle that allow progress to be judged more or less effectively. For example, software sizing technology can now predict the sizes of both specifications and the volume of source code needed. Defect estimating tools can predict the numbers of bugs or errors that might be encountered and discovered. Although such milestones are not perfect, they are better than the former approaches.

Project management is responsible for establishing milestones, monitoring their completion, and reporting truthfully on whether the milestones were successfully completed or encountered problems. When serious problems are encountered, it is necessary to correct the problems before reporting that the milestone has been completed.

Failing or delayed projects usually lack of serious milestone tracking. Activities are often reported as finished while work was still on going. Milestones on failing projects are usually dates on a calendar rather than completion and review of actual deliverables.

Delivering documents or code segments that are incomplete, contain errors, and cannot support downstream development work is not the way milestones are used by industry leaders.

Another aspect of milestone tracking among industry leaders is what happens when problems are reported or delays occur. The reaction is strong and immediate: corrective actions are planned, task forces assigned, and correction begins to occur. Among

laggards, on the other hand, problem reports may be ignored and very seldom do corrective actions occur.

In more than a dozen legal cases involving projects that failed or were never able to operate successfully, project tracking was inadequate in every case. Problems were either ignored or brushed aside, rather than being addressed and solved.

Because milestone tracking occurs throughout software development, it is the last line of defense against project failures and delays. Milestones should be established formally, and should be based on reviews, inspections, and tests of deliverables. Milestones should not be the dates that deliverables more or less were finished. Milestones should reflect the dates that finished deliverables were validated by means of inspections, testing, and quality assurance review.

An interesting form of project tracking has been developed by the Shoulders Corporation for keeping track of object-oriented projects. This method uses a 3-Dimensional model of software objects and classes using Styrofoam balls of various sizes that are connected by dowels to create a kind of mobile. The overall structure is kept in a visible location viewable by as many team members as possible. The mobile makes the status instantly visible to all viewers. Color coded ribbons indicate status of each component, with different colors indicated design complete, code complete, documentation complete, and testing complete (gold). There are also ribbons for possible problems or delays. This method provides almost instantaneous visibility of overall project status. The same method has been automated using a 3-D modeling package, but the physical structures are easier to see and have proven more useful on actual projects. The Shoulders Corporation method condenses a great deal of important information into a single visual representation that non-technical staff can readily understand.

Summary and Results

Successful software projects can result from nothing more than avoiding the more serious mistakes that lead to disaster: 1) Look at the actual results of similar projects; 2) Make planning and estimating formal activities; 3) Plan for and control creeping requirements; 4) Use formal inspections as milestones for tracking project progress; 5) Collect accurate measurement data during your current project, to use with future projects.

Overcoming the risks shown here is largely a matter of opposites, or doing the reverse of what the risk indicates. Thus a well-formed software project will create accurate estimates derived from empirical data and supported by automated tools for handling the critical path issues. Such estimates will be based on the actual capabilities of the development team, and will not be arbitrary creations derived without any rigor. The plans will specifically address the critical issues of change requests and quality control. In addition, monthly progress reports will also deal with these critical issues. Accurate progress reports are the last line of defense against failures.

Suggested Readings

Charette, Bob; Software Engineering Risk Analysis and Management; McGraw Hill, New York, NY; 1989.

Charette, Bob; Application Strategies for Risk Management; McGraw Hill, New York, NY; 1990.

DeMarco, Tom; Controlling Software Projects; Yourdon Press, New York; 1982; ISBN 0-917072-32-4; 284 pages.

Everett, Gerald D. and McLeod, Raymond; Software Testing – Testing Across the Entire Software Development Life Cycle; IEEE Press; 2007.

Ewusi-Mensah, Kwaku; Software Development Failures; MIT Press, Cambridge, MA; 2003; ISBN 0-26205072-2/276 pages.

Fernandini, Patricia L.; A Requirements Pattern; Addison Wesley, Boston, MA; 2002; ISBN 0-201-73826-0.

Flowers, Stephen; Software Failures: Management Failures; Amazing Stories and Cautionary Tales; John Wiley & Sons; 1996.

Galorath, Dan and Evans, Michael; Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves; Auerbach; Philadelphia, PA; 2006.

Garmus, David and Herron, David; Function Point Analysis – Measurement Practices for Successful Software Projects; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3; 363 pages.

Gibbs, T. Wayt; “Trends in Computing: Software’s Chronic Crisis”; Scientific American Magazine, 271(3), International edition; pp 72-81; September 1994.

Gilb, Tom and Graham, Dorothy; Software Inspection; Addison Wesley, Harlow UK; 1993; ISBN 10: 0-201-63181-4.

Glass, R.L.; Software Runaways: Lessons Learned from Massive Software Project Failures; Prentice Hall, Englewood Cliffs; 1998.

International Function Point Users Group (IFPUG); IT Measurement – Practical Advice from the Experts; Addison Wesley Longman, Boston, MA; 2002; ISBN 0-201-74158-X; 759 pages.

Johnson, James et al; The Chaos Report; The Standish Group, West Yarmouth, MA; 2000.

Jones, Capers; Applied Software Measurement; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 pages; 3rd edition due in the Spring of 2008.

Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.

Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press, Boston, MA; December 1995; 250 pages; ISBN 1-850-32804-8; 292 pages.

Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.

Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 2007; ISBN 13-978-0-07-148300-1.

Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA; ISBN 0-201-48542-7; 2000; 657 pages.

Jones, Capers; "Sizing Up Software," Scientific American Magazine, Volume 279, No. 6, December 1998; pages 104-111.

Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Software Productivity Research technical report; Narragansett, RI; 2007; 65 pages.

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

Pressman, Roger; Software Engineering – A Practitioner’s Approach; McGraw Hill, NY; 6th edition, 2005; ISBN 0-07-285318-2.

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

Robertson, Suzanne and Robertson, James; Requirements-Led Project Management; Addison Wesley, Boston, MA; 2005; ISBN 0-321-18062-3.

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

Yourdon, Ed; Death March - The Complete Software Developer’s Guide to Surviving “Mission Impossible” Projects; Prentice Hall PTR, Upper Saddle River, NJ; ISBN 0-13-748310-4; 1997; 218 pages.

Yourdon, Ed; Outsource: Competing in the Global Productivity Race; Prentice Hall PTR, Upper Saddle River, NJ; ISBN 0-13-147571-1; 2005; 251 pages.

Web Sites

Information Technology Metrics and Productivity Institute (ITMPI): www.ITMPI.org

International Software Benchmarking Standards Group (ISBSG): www.ISBSG.org

International Function Point Users Group (IFPUG): www.IFPUG.org

Software Engineering Institute (SEI): www.SEI.org

Software Productivity Research (SPR): www.SPR.com

Suggested Web Sites

<http://www.IASAhome.org> This is the web site for the non-profit International Association of Software Architects (IASA). Software architecture is the backbone of all large applications. Good architecture can lead to applications whose useful life expectancy is 20 years or more. Questionable architecture can lead to applications whose useful life expectancy is less than 10 years, coupled with increasing complex maintenance tasks and high defect levels. The IASA is working hard to improve both the concepts of architecture and the training of software architects via a modern and extensive curriculum.

<http://www.IIBA.org> This is the web site for the non-profit International Institute of Business Analysis. This institute deals with the important linkage between business knowledge and software that supports business operations. Among the topics of concern are the Business Analysis Body of Knowledge (BABOK), training of business analysts, and certification to achieve professional skills.

<http://www.IFPUG.org> This is the web site for the non-profit International Function Point Users Group. IFPUG is the largest software metrics association in the world, and the oldest association of function point users. This web site contains information about IFPUG function points themselves, and also citations to the literature dealing with function points. IFPUG also offers training in function point analysis and administers. IFPUG also administers a certification program for analysts who wish to become function point counters.

<http://www.ISBSG.org> This is the web site for the non-profit International Software Benchmark Standards Group. ISBSG, located in Australia, collects benchmark data on software projects throughout the world. The data is self-reported by companies using a standard questionnaire. About 4,000 projects comprise the ISBSG collection as of 2007, and the collection has been growing at a rate of about 500 projects per year. Most of the data is expressed in terms of IFPUG function point metrics, but some of the data is also expressed in terms of COSMIC function points, NESMA function points, Mark II function points, and several other function point variants. Fortunately the data in variant metrics is identified. It would be statistically invalid to include attempt to average IFPUG and COSMIC data, or to mix up any of the function point variations.

<http://www.iso.org> This is the web site for the International Organization for Standardization (ISO). The ISO is a non-profit organization that sponsors and publishes a variety of international standards. As of 2007 the ISO published about a thousand standards a year, and the total published to date is approximately 17,000. Many of the published standards affect software. These include the ISO 9000-9004 quality standards and the ISO standards for functional size measurement.

<http://www.PMI.org> This is the web site for the Project Management Institute (PMI). PMI is the largest association of managers in the world. PMI performs research and collects data on topics of interest to managers in every discipline: software, engineering, construction, and so forth. This data is assembled into the well known Project Management Body of Knowledge or PMBOK.

<http://www.ITMPI.org> This is the web site for the Information Technology Metrics and Productivity Institute. ITMPI is a wholly-owned subsidiary of Computer Aid Inc. The ITMPI web site is a useful portal into a broad range of measurement, management, and software engineering information. The ITMPI web site also provides useful links to many other web sites that contain topics of interest on software issues.

<http://www.sei.cmu.edu> This is the web site for the Software Engineering Institute (SEI). The SEI is a federally-sponsored non-profit organization located on the campus of Carnegie Mellon University in Pittsburgh, PA. The SEI carries out a number of research programs dealing with software maturity and capability levels, with quality, risks, measurement and metrics, and other topics of interest to the software community.

<http://www.stsc.hill.af.mil/CrossTalk> This is the web site of both the Air Force Software Technology Support Center (STSC) and also the CrossTalk journal, which is published by the STSC. The STSC gathers data and performs research into a wide variety of software engineering and software management issues. The CrossTalk journal is one of few technical journals that publish full-length technical articles of 4,000 words or more. Although the Air Force is the sponsor of STSC and CrossTalk, many topics are also relevant to the civilian community. Issues such as quality control, estimating, maintenance, measurement, and metrics have universal relevance.

