Minimizing the Risk of Litigation

Problems Noted in Breach of Contract Litigation

Capers Jones, Vice President and CTO, Namcook Analytics LLC

Abstract. While working as an expert witness in a number of lawsuits where large software projects were canceled or did not operate correctly when deployed, the author has observed six major problems that occur repeatedly: 1 — Accurate estimates are not produced or are overruled. 2 — Accurate estimates are not supported by defensible benchmarks. 3 — Requirements changes are not handled effectively. 4 — Quality control is deficient. 5 — Progress tracking fails to alert higher management to the seriousness of issues. 6 — Contracts omit important topics such as change control and quality, or include hazardous terms.

Introduction

Much of the software literature deals with "best practices." This article concentrates on "worst practices," or the factors that most often lead to software failure and litigation.

For the purposes of this article, software "failures" are defined as software projects that have any of these attributes:

1. Termination of the project due to cost or schedule overruns.
2. Sche dule or cost overruns in excess of 50 percent of initial estimates.
3. Applications that, upon deployment, fail to operate safely.
4. Law suits brought by clients for contractual noncompliance.

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

1. Accurate estimates were either not prepared or were rejected.
2. Accurate estimates were not supported by objective benchmarks.
3. Change control was not handled effectively.
4. Quality control was inadequate.
5. Progress tracking did not reveal the true status of the project.
6. The contracts omitted key topics, such as quality and out-of-scope changes

Readers are urged to discuss outsource agreements with their attorneys. This paper is based on observations of actual cases, but the author is not an attorney, and the paper is not legal advice. It is advice about how software projects might be improved to lower the probability of litigation occurring.

To begin the discussion of defenses against software litigation, let us consider the normal outcomes of 15 types of U.S. software projects. Table 1 shows the percentage of projects that are likely to be on time, late, or canceled without being completed at all due to excessive cost, schedule overruns or poor quality:

	Application Types	On-time	Late	Canceled
1	Scientific	68%	20%	12%
2	Smart phones	67%	19%	14%
3	Open source	63%	36%	7%
4	U.S. outsource	60%	30%	10%
5	Cloud	59%	29%	12%
6	Web applications	55%	30%	15%
7	Games and entertainment	54%	36%	10%
8	Offshore outsource	48%	37%	15%
9	Embedded software	47%	33%	20%
10	Systems and middleware	45%	45%	10%
11	Information technology (IT)	45%	40%	15%
12	Commercial	44%	41%	15%
13	Military and defense	40%	45%	15%
14	Legacy renovation	30%	55%	15%
15	Civilian government	27%	63%	10%
	Total Applications	**50.13%**	**37.27%**	**13%**

Table 1: Outcomes of U.S. software projects circa 2016

As can be seen, schedule delays and canceled projects are distressingly common among all forms of software this year. This explains why software is viewed by most CEOs as the least competent and least professional form of engineering in the current business world.

Note that the data in Table 1 is from benchmark and assessment studies carried out by the author and his colleagues between 1984 and 2016. Unfortunately, recent data since 2010 is not much better than older data before 1990. This is due to several factors, including the following: 1) very poor measurement practices and distressingly bad metrics, which prevent improvements from being widely known, and 2) software that continues to use custom designs and manual coding, both of which are intrinsically expensive and error prone. Until the software industry adopts modern manufacturing concepts that utilize standard reusable components instead of custom-built artifacts, software can never be truly cost effective.

Let us consider each of these six topics in turn.

Problem 1: Estimating Errors and Estimate Rejection

Although cost estimating is difficult, there are a number of commercial software parametric cost estimating tools that do a capable job. COCOMO III, CostXpert, ExcelerPlan, KnowledgePlan, True Price, SEER, SLIM and the author's Software Risk Master™ (SRM) are examples available in the United States.

Despite the proven accuracy of parametric estimation tools and their widespread availability, as of 2016, less than 20 percent of the author's clients used any formal estimating methods at all when we first carried out software process evaluation studies. It is alarming that 80 percent of U.S. software companies and projects in 2016 still lag in formal sizing and the use of parametric estimation tools.

However, even if accurate estimates can be produced using commercial parametric estimating tools, clients or executives may not accept them. In fact, about half of the cases the author observed during litigation did not produce accurate estimates at all and did not use parametric estimating tools. Manual estimates tend toward optimism, or predicting shorter schedules and lower costs than actually occur.

Problem 2: Missing Defensible Objective Benchmarks

Somewhat surprisingly, the other half of the cases in litigation had accurate parametric estimates that had been rejected and replaced by arbitrary forced "estimates" based on business needs rather than team abilities. These pseudo-estimates were not produced using parametric estimation tools but were arbitrary schedule demands by clients or top executives based on perceived business needs.

The main reason that the original accurate parametric estimates were rejected and replaced was the absence of supporting historical benchmark data. Without accurate 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.

Some foreign governments have improved contract accuracy by mandating function point metrics: the governments of Brazil, Japan, Malaysia, Mexico and Italy require function point size and cost information for all government software contracts. Eventually all governments will probably require function point metrics for contracts, but no doubt U.S. state governments and the U.S. federal government will be among the last to do this since they lag in so many other software disciplines. (The author has been an expert witness in more lawsuits involving state governments than any other industry. Government software problems are often national news, e.g., the delay of President Barack Obama's Affordable Care Act.)

Problem 3: Rapidly Changing Requirements

The average rate at which software requirements change has been measured to range from about 0.5 percent per calendar month to as high as 4 percent per calendar month. Thus, for a project with a 12-month schedule, more than 10 percent of the features in 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.

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

- Estimating the number and rate of development changes before starting.
- Using function point metrics to quantify changes.
- Using a joint client/development change control board or designated domain experts.
- Using model-based requirements methodologies.
- Calculating the FOG and Flesch readability indices of requirements.
- Involving full-time user representatives for Agile projects.
- Using joint application design (JAD) to minimize downstream changes.
- Using quality function deployment (QFD) for quality requirements.
- Training in requirements engineering for business analysts and designers.
- Using formal requirements inspections to minimize downstream changes.
- Using formal prototypes to minimize downstream changes.
- Planning usage of iterative development to accommodate changes.
- Formally reviewing all change requests.
- Revising cost and schedule estimates for all changes greater than 10 function points.
- Prioritizing change requests in terms of business impact.
- Formally assigning change requests to specific releases.
- Using automated change control tools with cross-reference capabilities.

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

Requirements changes will always occur for large systems. It is not possible to freeze the requirements of any real-world application, and it is naive 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.

Problem 4: Poor Quality Control

It is dismaying to observe the fact that two of the most effective technologies in all of software are almost never used on projects that turn out to be disasters and end up in court. First, 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.

Second, the technology of static analysis has been available since 1984 and has proven to be effective in finding code bugs rapidly and early (although static analysis does not find requirements, architecture and design problems).

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

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, and 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" (DRE) 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 before delivery and the users find 100 defects in a standard time period after release (normally 90 days), then the defect removal efficiency is 90 percent.

The author strongly recommends that defect removal efficiency levels (DRE) be included in all software outsource and development contracts, with 96 percent being a proposed minimum acceptable level of defect removal efficiency. For medical devices and weapons systems, a higher rate of about 99 percent defect removal efficiency should be written into the contracts.

(The U.S. average in 2016 is only about 92 percent. Agile projects average about 92 percent; waterfall are often below 85 percent. TSP and RUP are among the quality strong methods that usually top 96 percent in defect removal efficiency.)

A rate of 96 percent is a significant improvement over current norms. For some mission-critical applications, a higher level such as 99.8 percent might be required. It is technically challenging to achieve such high levels of defect removal efficiency, and it can't be done by testing alone.

In order to top 98 percent in defect removal efficiency, formal inspections and pre-test static analysis plus at least eight forms of testing are needed (1 — unit test; 2 — function test; 3 — regression test; 4 — component test; 5 — performance test; 6 — usability test; 7 — system test; 8 — acceptance or beta test).

Table 2 shows combinations of quality control factors that can lead to high, average or poor defect removal efficiency (DRE).

Successful projects in the 10,000 function point range accumulate development totals of around 4.0 defects per function point and remove about 98 percent of them before delivery to customers. In other words, the number of delivered defects is about 0.2 defects per function point, or 800 total latent defects. Of these, about 10 percent — or 80 — would be fairly serious defects. The rest would be minor or cosmetic defects. Stabilization, or the number of calendar months required to achieve safe operation of the application, would be about 2.5 months.

By contrast, the unsuccessful projects of 10,000 function points that end up in court accumulate development totals of around 6.0 defects per function point and remove only about 85 percent of them before delivery. The number of delivered defects is about 0.9 defects per function point, or 9,000 total latent defects. Of these, about 15 percent — or 1,350 — 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. Stabilization, or the number of calendar months required to achieve safe operation of the application, might stretch out to 18 months or more.

Unsuccessful projects typically omit design and code inspections and static analysis and depend solely on testing. The omission of upfront inspections and static analysis causes four serious problems: 1) The large number of defects still present when testing begins slows 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 testing only is not sufficient to achieve defect removal rates higher than about 85 percent; and 4) Applications that bypass both inspections and static analysis have a strong tendency to include error-prone modules.

	Defect Removal Efficiency (DRE)	> 99 %	95%	< 87%
1	Formal requirement inspections	Yes	No	No
2	Formal design inspections	Yes	No	No
3	Formal code inspections	Yes	No	No
4	Formal security inspections	Yes	No	No
5	Static analysis	Yes	Yes	No
6	Unit test	Yes	Yes	Yes
7	Function test	Yes	Yes	Yes
8	Regression test	Yes	Yes	Yes
9	Integration test	Yes	Yes	Yes
10	Usability test	Yes	Yes	No
11	Security test	Yes	Yes	No
12	System test	Yes	Yes	Yes
13	Acceptance test		Yes	Yes

Table 2: Ranges of DRE for 1,000 function point applications

Problem 5: Poor Software Milestone Tracking

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; and 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.

There are also "non-functional requirements," often due to outside influences. These can change abruptly and many are not under control of software groups. For example, a change in federal or state laws may require changes to hundreds of applications, including some that are under development.

As of 2016, 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 issues or problems.

Examples of tracking tools include Automated Project Office (APO), Microsoft project management suite, OmniTracker, Capterra, and perhaps 50 others with various capabilities. However, in spite of the availability of these tools, less than 45 percent of the author's clients in our initial process evaluation studies used any of them.

For an industry now more than 65 years of age, it is somewhat surprising that there is no general or universal set of project milestones for indicating tangible progress. From the author'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 or inspection 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.

The most important aspect of Table 3 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. These “red flag” topics are those that are likely to cause schedule delays, cost overruns or both.

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 in 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:

1. Status of last month’s “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 month’s 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

Table 4: Suggested format for monthly status reports for software projects

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 affect project milestone completions, and this kind of situation cannot

1. Application sizing completed using both function points and code statements
2. Application risk predictions completed
3. Application size and risk predictions reviewed
4. Requirements document completed
5. Requirements document inspection completed
6. Initial cost estimate completed
7. Initial cost estimate review completed
8. Development plan completed
9. Development plan review completed
10. Cost tracking system initialized
11. Defect tracking system initialized
12. Prototype completed
13. Prototype review completed
14. Complexity analysis of base system (for enhancement projects)
15. Code restructuring of base system (for enhancement projects)
16. Functional specification completed
17. Functional specification review completed
18. Data specification completed
19. Data specification review completed
20. Logic specification completed
21. Logic specification review completed
22. Quality control plan completed
23. Quality control plan review completed
24. Change control plan completed
25. Change control plan review completed
26. Security plan completed
27. Security plan review completed
28. User information plan completed
29. User information plan review completed
30. Code for specific modules completed
31. Code inspection for specific modules completed
32. Code for specific modules unit tested
33. Test plan completed
34. Test plan review completed
35. Test cases for specific test stage completed
36. Test case inspection for specific test stage completed
37. Test stage completed
38. Test stage review completed
39. Integration for specific build completed
40. Integration review for specific build completed
41. User information completed
42. User information review completed
43. Quality assurance sign off completed
44. Delivery to beta test clients completed
45. Delivery to clients completed

Table 3: Representative tracking milestones for large software projects

be anticipated. The same is true of natural phenomena such as hurricanes or earthquakes, which can shut down businesses.

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 both the sizes of specifications and the volume of source code needed. Defect estimating tools can predict the number of bugs or errors that might be encountered and discovered. Although such milestones are not perfect, they are better than former approaches.

Project management is responsible for establishing milestones, monitoring their completion, and reporting truthfully 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 milestones have been completed.

Failing or delayed projects usually lack serious milestone tracking. Activities are often reported as finished while work is still ongoing. 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.

In more than a dozen legal cases involving projects that failed or were never able to operate successfully, project tracking was inadequate. 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 were more or less finished. Milestones should reflect the dates that finished deliverables were validated by means of inspections, testing and quality assurance review.

Problem 6: Flawed Outsource Agreements That Omit Key Topics

In several of the cases where the author has been an expert witness, the contracts themselves seemed flawed and omitted key topics that should have been included. Worse, some contracts included topics that probably should have been omitted. Here are some examples:

• In one case the contract required that the software delivered by the vendor should have "zero defects." Since the application approached 10,000 function points in size, zero-defect software is beyond the current state of the art. The software as delivered did not have very many defects and, in fact, was much better than average. But it was not zero-defect software, and hence the vendor was sued.

• A fixed-price contract had clauses for "out of scope" requirements changes. In this case, the client unilaterally added 82 major changes totaling about 3,000 new function points. But the contract did not define the phrase "out of scope," and the client asserted that the changes were merely elaborations to existing requirements and did not want to pay for them.

• In another fixed-price contract the vendor added about 5,000 function points of new features very late in development. Here the client was willing to pay for the added features. However, features added after design and during coding are more expensive to build than features during normal development. In this case the vendor was asking for additional payments to cover the approximate 15 percent increase in costs for the late features. Needless to say, there should be a sliding scale of costs that goes up for features added three, six, nine, 12 or more months after the initial requirements are defined and approved by the client. The fee structure might be something like an increase of 3 percent, 5 percent, 7 percent, 12 percent and 15 percent based on calendar month intervals.

—In several contracts where the plaintiff alleged poor quality on the part of the vendor, the contracts did not have any clauses that specified acceptable quality, such as defect removal efficiency (DRE) or maximum numbers of bugs found during an acceptance test. In the absence of any contractual definitions of "poor quality," such charges are difficult to prove.

The bottom line is that clients, vendors and their attorneys should be sure that all outsource contracts include clauses dealing with requirements changes, quality, and delivered defects, and also penalties for schedule delays caused by vendor actions.

Note that the author is not an attorney and this is not legal advice. But it is obvious that every software outsource contract should include clauses for quality and for requirements changes, especially late requirements changes. Attorneys should be involved in structuring the proper clauses in software outsource agreements.

Summary and Observations Based on Breach of Contract Litigation

Successful software projects can result from nothing more than avoiding the more serious mistakes that lead to disaster. A set of basic steps can lower the odds of a failing project and litigation: 1) Use parametric estimation tools and avoid manual estimates; 2) Look at the actual benchmark results of similar projects; 3) Make planning and estimating formal activities; 4) Plan for and control creeping requirements; 5) Use formal inspections as milestones for tracking project progress; 6) Include pre-test static analysis and inspections in quality control; 7) Collect accurate measurement data during your current project to use with future projects; 8) Ensure with your attorneys that contracts have suitable clauses for requirements growth and quality levels of delivered materials. Omitting these two topics can lead to very expensive litigation later.

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

1. Abrain, A. (2015.) Software Estimating Models. Wiley-IEEE Computer Society.
2. Abrain, A. (2010.) Software Metrics and Metrology. Wiley-IEEE Computer Society.
3. Abrain, A. (2008.) Software Maintenance Management: Evolution and Continuous Improvement. Wiley-IEEE Computer Society.
4. Beck, K. (2002.) Test-Driven Development. Addison Wesley, Boston, Mass. ISBN 10: 0321146530, 240 pages.
5. Black, R. (2009.) Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Wiley. ISBN-10 0470404159, 672 pages.
6. Boehm, B. (1981.) Software Engineering Economics. Prentice Hall, Englewood Cliffs, N.J., 900 pages.
7. Brooks, F. (1974, rev. 1995.) The Mythical Man-Month. Addison-Wesley, Reading, Mass.
8. Bundschuh, M. & Dekkers, C. (2005.) The IT Metrics Compendium. Springer.
9. Charette, B. (1989.) Software Engineering Risk Analysis and Management. McGraw-Hill, New York, N.Y.
10. Charette, B. (1990.) Application Strategies for Risk Management. McGraw-Hill, New York, N.Y.
11. DeMarco, T. (1982.) Controlling Software Projects. Yourdon Press, N.Y. ISBN 0-917072-32-4, 284 pages.
12. Ebert, C., Dumke, R. & Bundschuh M. (2004.) Best Practices in Software Measurement. Springer.
13. Everett, G. D. & McLeod, R. (2007.) Software Testing – Testing Across the Entire Software Development Life Cycle. IEEE Press.
14. Ewusi-Mensah, K. (2004.) Software Development Failures. MIT Press, Cambridge, Mass. ISBN 0-26205072-2276.
15. Fernandini, P. L. (2002.) A Requirements Pattern. Addison Wesley, Boston, Mass. ISBN 0-201-73826-0.
16. Flowers, S. (1996.) Software Failures: Management Failures; Amazing Stories and Cautionary Tales. John Wiley & Sons.
17. Gack, G. (2010.) Managing the Black Hole: The Executive's Guide to Software Project Risk. Business Expert Publishing, Thomson, Georgia. ISBN10: 1-935602-01-9.
18. Galorath, D. & Evans, M. (2006.) Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves. Auerbach; Philadelphia, Penn.
19. Garmus, D. & Herron, D. (2001.) Function Point Analysis – Measurement Practices for Successful Software Projects. Addison Wesley Longman, Boston, Mass. ISBN 0-201-69944-3, 363 pages.
20. Garmus, D. & Herron, D. (1995.) Measuring the Software Process: A Practical Guide to Functional Measurement. Prentice Hall, Englewood Cliffs, N.J.
21. Garmus, D., Russac, J. & Edwards, R. (2010.) Certified Function Point Counters Examination Guide. CRC Press.
22. Glass, R.L. (1998.) Software Runaways: Lessons Learned from Massive Software Project Failures. Prentice Hall, Englewood Cliffs.
23. Gibbs, T. W. (1994, September.) Trends in Computing: Software's Chronic Crisis. Scientific American Magazine, 271(3), International edition, 72-81.
24. Gilb, T. & Graham, D. (1993.) Software Inspection. Addison Wesley, Harlow, U.K. ISBN 10: 0-201-63181-4.
25. Glass, R.L. (1998.) Software Runaways: Lessons Learned from Massive Software Project Failures. Prentice Hall, Englewood Cliffs.
26. Harris, M. D. S., Herron, D. & Iwanicki, S. (2008.) The Business Value of IT. CRC Press, Auerbach; Boca Raton, Fla. ISBN 978-14200-6474-2.
27. Hill, P., Jones, C. & Reifer, D. (2013 September.) The Impact of Software Size on Productivity. International Software Standards Benchmark Group (ISBSG), Melbourne, Australia.
28. International Function Point Users Group (IFPUG). (2002.) IT Measurement – Practical Advice from the Experts. Addison Wesley Longman, Boston, Mass. ISBN 0-201-74158-X, 759 pages.
29. Johnson, J. et al. (2000.) The Chaos Report. The Standish Group, West Yarmouth, Mass.
30. Jones, C. (2015.) The Technical and Social History of Software Engineering. Addison Wesley (contains summaries of important software industry lawsuits such as anti-trust and patent violations).
31. Jones, C. (2012.) Studio 38 in Rhode Island – A Study of Software Risks. Published in various Rhode Island newspapers such as "The Providence Journal," "South County Independent," "Narragansett Times," etc.
32. Jones, C. & Bonsignour, O. (2011.) The Economics of Software Quality. Addison Wesley, Boston, Mass. ISBN 10 0-13-258220-1, 587 pages.
33. Jones, C. (2010.) Software Engineering Best Practices. McGraw Hill, New York, N.Y. ISBN 978-0-07-162161-8, 660 pages.
34. Jones, C. (2008.) Applied Software Measurement. McGraw Hill, 3rd edition. ISBN 978-0-07-150244-3, 662 pages.
35. Jones, C. (1994.) Assessment and Control of Software Risks. Prentice Hall. ISBN 0-13-741406-4, 711 pages.
36. Jones, C. (1995, December.) Patterns of Software System Failure and Success. International Thomson Computer Press, Boston, Mass. 250 pages. ISBN 1-850-32804-8, 292 pages.
37. Jones, C. (1997.) Software Quality – Analysis and Guidelines for Success. International Thomson Computer Press, Boston, Mass. ISBN 1-85032-876-6, 492 pages.
38. Jones, C. (2007.) Estimating Software Costs. McGraw Hill, New York, N.Y. ISBN 13-978-0-07-148300-1.
39. Jones, C. (2000.) Software Assessments, Benchmarks, and Best Practices. Addison Wesley Longman, Boston, Mass. ISBN 0-201-48542-7, 657 pages.
40. Jones, C. (1998, December.) Sizing Up Software. Scientific American Magazine, Vol. 279, No. 6, 104–111.
41. Jones, C. (2007.) Conflict and Litigation Between Software Clients and Developers; Software Productivity Research technical report. Narragansett, R.I., 65 pages.
42. Kan, S. H. (2003.) Metrics and Models in Software Quality Engineering, 2nd edition. Addison Wesley Longman, Boston, Mass. ISBN 0-201-72915-6, 528 pages.
43. Pressman, R. (2005.) Software Engineering – A Practitioner's Approach. McGraw Hill, N.Y. 6th edition. ISBN 0-07-285318-2.
44. Radice, R. A. (2002.) High Quality Low Cost Software Inspections. Paradoxicon Publishing, Andover, Mass. ISBN 0-9645913-1-6, 479 pages.
45. Robertson, S. & Robertson, J. (2005.) Requirements-Led Project Management. Addison Wesley, Boston, Mass. ISBN 0-321-18062-3.
46. Wiegers, K. E. (2002.) Peer Reviews in Software – A Practical Guide. Addison Wesley Longman, Boston, Mass. ISBN 0-201-73485-0, 232 pages.
47. Yourdon, E. (1997.) Death March - The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects. Prentice Hall PTR, Upper Saddle River, N.J. ISBN 0-13-748310-4, 218 pages.
48. Yourdon, E. (2005.) Outsource: Competing in the Global Productivity Race. Prentice Hall PTR, Upper Saddle River, N.J. ISBN 0-13-147571-1, 251 pages.

Websites

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
Namcook Analytics LLC: www.Namcook.com
Namcook Analytics Blog: http://NamcookAnalytics.com
Reifer Consulting: www.Reifer.com
Software Engineering Institute (SEI): www.SEI.cmu.edu
Software Productivity Research (SPR): www.SPR.com

SUGGESTED WEBSITES

http://www.IASAhome.org
This is the website for the nonprofit International Association of Software Architects (IASA). Software architecture is the backbone of all large applications. Good architecture can lead to applications with useful life expectancies of 20 years or more. Questionable architecture can lead to applications with useful life expectancies of fewer 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 website for the nonprofit International Institute of Business Analysis. This institute deals with the important link 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 website for the nonprofit International Function Point Users Group. IFPUG is the largest software metrics association in the world and the oldest association of function point users. This website contains information about IFPUG function points themselves and also includes 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.ITMPI.org
This is the website for the Information Technology Metrics and Productivity Institute. ITMPI is a wholly owned subsidiary of Computer Aid Inc. The ITMPI website is a useful portal into a broad range of measurement, management and software engineering information. The ITMPI website also provides useful links to many other websites that contain topics of interest on software issues.

http://www.ISBSG.org
This is the website for the nonprofit 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 attempts to average IFPUG and COSMIC data, or to mix up any of the function point variations.

http://www.iso.org
This is the website for the International Organization for Standardization (ISO). The ISO is a nonprofit organization that sponsors and publishes a variety of international standards. As of 2007 the ISO published about a thousand standards per 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.namcook.com
This website contains a variety of quantitative reports on software quality and risk factors. It also contains a patented high-speed sizing tool that can size applications of any size in 90 seconds or fewer. It also contains a catalog of software benchmark providers that currently lists 20 organizations that provide quantitative data about software schedules, costs, quality and risks.

http://www.PMI.org
This is the website 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.sei.cmu.edu
This is the website for the Software Engineering Institute (SEI). The SEI is a federally sponsored nonprofit organization located on the campus of Carnegie Mellon University in Pittsburgh, Penn. 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 website 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.

ABOUT THE AUTHOR

Capers Jones is currently the President and CEO of Capers Jones & Associates LLC. He is also the founder and former chairman of Software Productivity Research LLC (SPR). He holds the title of Chief Scientist Emeritus at SPR. Capers Jones founded SPR in 1984.

Before founding SPR, Capers was Assistant Director of Programming Technology for the ITT Corporation at the Programming Technology Center in Stratford, Conn. He was also a manager and researcher at IBM in California.

Capers is a well-known author and international public speaker. Some of his books have been translated into six languages. All of his books are translated into Japanese and his newest books are available in Chinese editions as well.

www.Namcook.com
http://namcookanalytics.com
Capers.Jones3@gmail.com