MINIMIZING THE RISK OF LITIGATION:

PROBLEMS NOTED IN BREACH OF CONTRACT LITIGATION

Draft 20.0 – April 9, 2016

Keywords

Breach of contract litigation for software, Capers Jones data, joint benchmarks, software cost estimation, ISBSG, Namcook Analytics, parametric estimation, software progress tracking, software requirements creep, technical debt.

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, six major problems 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 the issues; 6) Contracts themselves omit important topics such as change control and quality, or include hazardous terms.

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. This article discusses the topics that should be included in software project tracking reports to minimize failures, delays, and costly litigation.

Capers Jones, Vice President and CTO, Namcook Analytics LLC
Web: www.Namcook.com
Blog: http://namcookanalytics.com
Email: Capers.Jones3@gmail.com

MINIMIZING THE RISK OF LITIGATION:
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 end 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 the 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 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 odds of litigation occurring.

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

Table 1: Outcomes of U.S. Software Projects Circa 2016

	Application Types	**On-time**	**Late**	**Canceled**
1	Scientific	68.00%	20.00%	12.00%
2	Smart phones	67.00%	19.00%	14.00%
3	Open source	63.00%	36.00%	7.00%
4	U.S. outsource	60.00%	30.00%	10.00%
5	Cloud	59.00%	29.00%	12.00%
6	Web applications	55.00%	30.00%	15.00%
7	Games and entertainment	54.00%	36.00%	10.00%
8	Offshore outsource	48.00%	37.00%	15.00%
9	Embedded software	47.00%	33.00%	20.00%
10	Systems and middleware	45.00%	45.00%	10.00%
11	Information technology (IT)	45.00%	40.00%	15.00%
12	Commercial	44.00%	41.00%	15.00%
13	Military and defense	40.00%	45.00%	15.00%
14	Legacy renovation	30.00%	55.00%	15.00%
15	Civilian government	27.00%	63.00%	10.00%
	Total Applications	**50.13%**	**37.27%**	**13.00%**

As can be seen schedule delays and cancelled projects are distressingly common among all forms of software in 2016. This explains why software is viewed by most CEO's as the least competent and least professional form of engineering of the current business world.

Note that the data in table 1 is from benchmark and assessment studies carried out by the author and colleagues between 1984 and 2016. Unfortunately recent data since 2010 is not much better than older data before 1990. This is due to very poor measurement practices and distressingly bad metrics which prevent improvements from being widely known.

Schedule delays unfortunately get larger with application size as shown by the approximate results in table 2:

Table 2: Planned versus Actual Schedules

Function Points	Planned Schedule Months	Probable Schedule Months	Difference	Percent of Plan
10	2.40	2.51	0.11	4.71%
100	5.75	6.31	0.56	9.65%
1000	13.80	15.85	2.05	14.82%
10000	33.11	39.81	6.70	20.23%
100000	79.43	100.00	20.57	25.89%

Most of the lawsuits where the author worked as an expert witness were larger than 10,000 function points and were more than 18 months or 40% late when the projects were terminated and litigation was filed. Such long schedule delays keep accumulating expenses and eventually cause negative returns on investment (ROI) which leads to cancellation and sometimes to litigation as well.

For reasons outside the scope of this paper government software projects have much greater risks than civilian projects of the same size. The author has been an expert witness in more lawsuits for failing state government projects than for any other industry.

Another major reason for delays and cancelled projects is the fact that software 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.

Table 3 shows the risk patterns associated with large systems in the 10,000 function point size range:

Table 3: Risk Patterns for 10,000 Function Point Software Systems

	Risk Patterns	**Risk Percent**
1	**Odds of optimistic manual schedule estimates =**	**85.32%**
2	**Odds of inaccurate status tracking =**	**66.55%**
3	**Odds of project cancellation =**	**37.59%**
4	**Odds of toxic requirements that should not be included =**	**20.94%**
5	**Odds of feature bloat and unused features =**	**39.81%**
6	**Odds of outsource litigation =**	**20.72%**
7	**Odds of negative Return on Investment (ROI) =**	**51.49%**
8	**Odds of cost overrun =**	**29.77%**
9	**Odds of schedule delays =**	**36.99%**
10	**Odds of deferred features =**	**33.38%**
11	**Odds of high maintenance costs =**	**41.16%**
12	**Odds of poor customer satisfaction =**	**40.22%**
13	**Odds of poor executive satisfaction =**	**45.85%**
14	**Odds of poor team morale =**	**29.81%**
15	**Odds of post-release cyber-attacks =**	**17.93%**
	Average of all risks =	**39.83%**

Having looked at the problems of software, and large systems in particular, let us consider each of the six topics that cause litigation 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.

As a class the parametric estimation tools are more accurate than manual estimating methods. This is especially true for large systems > 10,000 function points in size. Almost every breach of contract case where the author has been an expert witness has been a large application > 10,000 function points in size.

In spite of the proven accuracy of parametric estimation tools and widespread availability, as of 2016 less than 20% 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% of U.S. software companies and projects in 2016 still lag in formal sizing and the use of parametric estimation tools.

However just because an accurate estimate can be produced using a commercial parametric 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 parametric estimating tools. Manual estimates tend towards optimism or predicting shorter schedules and lower costs than actually occur.

However many projects that did use parametric tools and where the estimates were later shown to be accurate had the parametric estimates rejected by clients of executives, for reasons discussed in the next section of this report/

Based on 50 samples of each, manual estimates and parametric estimates produced similar results below 250 function points. However as sizes increased manual estimates became progressively optimistic and understated both costs and schedules by more than 25% above 5,000 function points.

Early estimates using parametric estimation tools combined with early risk analyses are a solid first-line defense against later litigation. The author's Namcook estimation tool Software Risk Master ™ (SRM) includes patent-pending features for early estimation prior to requirements and also includes an integral risk analysis feature. It predicts both the odds and costs of breach of contract litigation as well. SRM can create risk, cost, and schedule estimates 30 to 180 days earlier than other estimating tools and estimating methods due to the patent-pending early sizing method.

Problem 2: Missing Defensible Objective Benchmarks

Somewhat surprisingly, the other half of the cases in litigation had accurate parametric estimates, but these estimates were 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.

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 parametric 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 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.

This problem has occurred in many lawsuits, and is particularly common in government software applications at the state and Federal levels. The delays in the California child support application, the Rhode Island motor vehicle application, Obamacare and the major FBI application show this, as do the more recent British government software delays. The many lawsuits involving delayed or canceled state government applications are a chronic problem for U.S. state governments and are also common for Federal civilian projects, and to a lesser degree for Federal defense contracts.

The author has been an expert witness in more failing state government software projects than all other industries combined. Indeed the author's home state of Rhode Island has experienced a recent major software delay in a motor vehicle application due in part to optimistic estimates combined with poor quality and change control. Very poor status tracking was also a factor.

Worse Rhode Island had a $100,000,000 failure due to funding Studio 38 with zero due diligence and no effective risk analysis prior to funding. The author's Software Risk Master (SRM) tool predicted an 88% chance of failure for this project, although this was a retrospective prediction made after bankruptcy and litigation had already occurred.

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 5,000 projects are available, and new projects are added at a rate of perhaps 500 per year.

Other companies such as Namcook Analytics LLC, Reifer Consulting, Software Productivity Research (SPR), and the Quality/Productivity Management Group (QPMG), Galorath, Quantitative Software Management (QSM), Process Fusion, and the David's Consulting Group also provide quantitative benchmarks.

However, much of the available benchmark data is made available only on a subscription basis to specific clients of the organizations. The ISBSG data, by contrast, is available to the general public although there are fees. The Reifer data is also available commercially. Much of the author's data has been published in 17 books and several hundred journal articles, such as this one.

***Note**: in September of 2013 a joint benchmark report was published by Peter Hill of ISBSG, Capers Jones of Namcook Analytics, and by Don Reifer of Reifer Consulting. The title of the report is "The Impact of Software Size on Productivity." This report is available from the ISBSG web site, www.ISBSG.org for the general ISBSG web site or http://www.isbsg.com/collections/analysis-reports?page=2 for the actual report itself.*

Unfortunately state and national government organizations are much less likely to create accurate benchmarks than public and private civilian corporations. (Military software is usually better done than civilian government software due in part to mandates that CMMI level 3 be part of all defense software contracts.)

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 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 report deals with the overall impact of applications with sizes from below 100 function points up to about 100,000 function points. IT applications, systems software, web applications, and other types of projects are discussed.

Problem 3: Rapidly Changing Requirements

The average rate at which software requirements change is has been measured to range between about 1% per calendar month and as high as 4% per calendar month. Thus for a project with a 12 month schedule, more than 10% 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.

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%. A Canadian lawsuit dealt with a project that doubled its size in function points due to requirements creep. A recent arbitration in 2011 in Hong Kong dealt with a project that went from 15,000 to more than 20,000 function points at a rate of change that approached 5% per calendar month.

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. Indeed the governments of Brazil and South Korea now require function point metrics for all government software contracts and Japan is about to use function points for its government contracts..

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
- Using high-speed function point sizing on all changes
- A joint client/development change control board or designated domain experts
- Model-based requirements methodologies
- Running text-based static analysis tools against text requirements
- Calculating the FOG and Flesch readability indices of requirements
- Full time involvement by user representatives for Agile projects
- Use of joint application design (JAD) to minimize downstream changes
- Training in requirements engineering for business analysts and designers
- 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 3,000 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 4% 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 Windows 10) 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.

Several of the new model-based requirements methods are very successful in eliminating requirements defects. Some are so fast that they outpace requirements creep.

A new kind of static analysis tool that finds errors in text requirements is also a useful approach to minimizing requirements issues. This kind of tool can be paired with older tools that calculate the readability indexes of text documents; i.e. the FOG and Flesch readability indices.

New kinds of rapid-sizing tools such as Software Risk Master™ (SRM) can now predict both function points and logical code statements in about 90 seconds which is fast enough to allow requirements changes to be sized immediately without lengthy delays.

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.

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.

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.)

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

Static analysis is seldom used in projects that end up in court for breach of contract, even though it is common among best in class organizations. However there is a caveat: the software industry has over 2,700 programming languages as of 2015. Static analysis tool collectively only support about 25 of these languages as of 2015. Of course the common languages in 2015 such as Java, C++, C#, SQL etc. are supported as are some older languages such as COBOL and PL/I.

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.

Most failing projects seem to be on schedule until testing starts, when excessive defect volumes stretch out the test period by several hundred percent compared to the planned schedule.

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” (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 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 (DRE) be included in all software outsource and development contracts, with 96% being a proposed minimum acceptable level of defect removal efficiency. For medical devices and weapons systems a higher rate of about 99% defect removal efficiency should be written in to the contracts.

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

A rate of 96% is a significant improvement over current norms. For some mission-critical applications, a higher level such as 99.8% 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.

Formal inspections and pre-test static analysis plus at least 8 forms of testing are needed to top 98% in defect removal efficiency (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.)

In addition the use of mathematical test case design such as using design of experiments plus having certified test personnel are correlated with high levels of test defect removal efficiency. Automated testing and of course good automated test library controls are also needed for high DRE levels.

Following are some of the methods for defect prevention, pre-test defect removal, and testing that are associated with high quality levels and high defect removal efficiency levels:

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 or 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)
- Utilization of the new SEMAT approach

Pre-Test Defect Removal

- Requirements inspections
- Requirements static analysis
- Design inspections
- Document inspections (user's guides, tutorials, etc.)
- Text static analysis
- Code static analysis
- Code inspections
- Test plan and test case inspection
- Defect repair inspection
- Software quality assurance reviews

Test Defect Removal and Refactoring

- Unit testing (manual and automated)
- Component testing
- New function testing
- Regression testing (manual and automated)
- Performance testing
- Refactoring
- Usability testing
- Security testing
- System testing
- External Beta 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.

Table 4 shows the combinations of factors that can lead to high-efficiency defect removal compared to low-efficiency defect removal for applications of a nominal 1000 function points in size:

Table 4: Ranges of DRE for 1000 Function Point Applications

	Defect Removal Efficiency (DRE)	**> 99 %**	**95%**	**< 90%**
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	Yes

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 98% 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% or 80 would be fairly serious defects. The rest would be minor or cosmetic defects. Stabilization or the number of calendar months to achieve safe operation of the application would be about 2.5 months.

By contrast, the unsuccessful projects that end up in court accumulate development totals of around 6.0 defects per function point and remove only about 85% 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% 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 month to achieve safe operation of the application might stretch out to 18 months or more.

If these low-quality applications contain “error prone modules” with very high defect densities, stabilization may be impossible. Error-prone modules are often so complex and difficult to fix safely that they may need surgical removal and complete replacement before stable operation is possible.

Unsuccessful projects typically omit design and code inspections and static analysis, and depend purely on testing. The omission of up-front inspections causes four 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 85%; 4) Applications that bypass both inspections and static analysis have a strong tendency to include error-prone modules.

(Studies by IBM and other leading companies noted that bugs are not randomly distributed in large systems. They tend to clump in a small number of very buggy modules. In one case, there were 425 modules in a major data base application. Of these 300 were zero-defect modules with no customer-reported bugs. About 57% of the entire volume of reported defects were against only 31 modules out of the total of 425.)

Problem 5: Poor 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 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 also issues or problems.

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

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.

Table 6: Representative Tracking Milestones for Large Software Projects

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

The most important aspect of table 6 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:

Table 7: Suggested Format for Monthly Status Reports for Software Projects

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 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.

Another interesting form of project tracking has been developed by Computer Aid, Inc. This is an automated status tracking tool called Automated Project Office (APO) which collects continuous data throughout development and produces useful dash board displays of current status that are available to managers and technical contributors.

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 samples:

- 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% increase in costs for the late features. Needless to say there should be a sliding scale of costs that goes up for features added 3, 6, 9, 12, or more months after the initial requirements are defined and approved by the client. The fee structure might be something like increase by 3%, 5%, 7% 12%, and 15% 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 acceptance test. In the absence of any contractual definitions of "poor quality" such charges are difficult to prove.

The bottom line is that clients, vendor, and their attorneys should be sure that all outsource contracts include clauses dealing with requirements changes, quality, 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.

The High Costs and Business Interruptions Caused by Litigation

Breach of contract litigation is an expensive business activity and also one that requires hundreds of hours of executive time, thousands of hours of technical staff time, and hundreds to thousands of billable hours by litigation attorneys and by paralegal and support personnel.

From noting the high costs for both the plaintiffs and defendants in breach of contract litigation, the Namcook Analytics Software Risk Master ™ (SRM) tool includes a standard feature for predicting breach of contract costs for both the plaintiff and defendant. The costs assume the case goes through trial. Out of court settlements are random and unpredictable. A sample prediction for a breach of contract case for a 10,000 function point systems software project is shown below:

Table 8: Software Risk Master (SRM) Breach of Contract Predictions

(10,000 function point systems software)

1	Client experience level:	Inexperienced
2	Development team experience level:	Inexperienced
3	Methodology:	Waterfall
4	Language:	Java
5	KLOC	533
6	Pre-test inspections used?	No
7	Pre-test static analysis used?	No
8	Formal test case design used?	No
9	Certified test personnel used?	No
10	Defect removal efficiency	< 90%
11	Delivered defects	8,500
12	High-severity delivered defects	1,105
13	Delivered defects per function point	0.85
14	Delivered defects per KLOC	15.95

15	High-severity per function point	0.11	
16	High severity per KLOC	2.07	
17	Technical debt for application	$2,656,250	
18	Probability of Litigation	22.50%	
19	Planned project schedule:	24 months	
20	Actual project schedule:	32 months	
21	Schedule slip:	8 months	
22	Litigation filed:	35 months	
23	Probable trial duration:	26 months	
24	Odds of out of court settlement:	76%	
25	Planned project cost		$18,900,000
26	Project cost at delivery		$25,515,000
27	Cost overrun		**$6,615,000**
28	Plaintiff executive hourly costs		$500
29	Plaintiff staff hourly costs		$100
30	Plaintiff expert witness hourly costs		$475
31	Plaintiff attorney hourly costs		$450
32	Plaintiff paralegal hourly costs		$200
33	Plaintiff consequential damages		**$11,985,250**
34	Defendant executive hourly costs		$500
35	Defendant staff hourly costs		$100
36	Defendant expert witness hourly costs		$475
37	Defendant attorney hourly costs		$450
38	Defendant paralegal hourly costs		$200
39	Plaintiff executive costs		$487,500
40	Plaintiff staff costs		$195,000
41	Plaintiff expert witness costs		$230,375
42	Plaintiff attorney costs		$1,030,500
43	Plaintiff paralegal costs		$705,000
		PLAINTIFF TOTAL	**$2,648,375**
		Total per function point	$264.84
44	Defendant executive costs		$541,125
45	Defendant staff costs		$220,350
46	Defendant expert witness costs		$256,868

47	Defendant attorney costs		$1,156,221
48	Defendant paralegal costs		$796,650
		DEFENDANT TOTAL	**$2,971,214**
		Total per function point	$297.12
49	Possible Damage Award		$15,000,000
50	Total Court Costs (both sides)		$5,619,589
51	Maximum cost if defendant loses		**$20,619,589**
		Total cost per function point	$2,061.96
52	Technical debt % of total costs		12.88%
53	Maximum cost if plaintiff loses		**$5,619,589**

As can be seen the nominal cost for the entire project was only $18,900,000. Yet the potential costs to the defendant if the case is lost might be $20,619,589. The high costs of litigation make clear the need for both excellence in outsource contracts and also professionalism in software development methods.

Note also that the “technical debt” for the bugs that caused the litigation amount to only about 13% of the possible costs to the defendant should the defendant lose the case. One of the issues with the technical debt metaphor is that it does not cover all of the costs of poor quality, and omits both litigation costs and also damages. For that matter technical debt also omits consequential damages for the plaintiff, or the actual harm caused by the bugs.

Table 8 only deals with breach of contract litigation. Software projects are unfortunately also susceptible to several other kinds of litigation including but not limited to:

- Patent violations from patent trolls
- Patent violations from legitimate patent holders
- Litigation for bias in civilian and defense contract awards
- Theft of algorithms and code from business competitors
- Non-competition and employment agreement issues
- Fraud charges from dissatisfied clients
- Possible damages from harm done by software in cases such as brake failures
- Possible criminal charges for Sarbanes-Oxley violations

Attorneys and legal costs are a steadily increasing source of expense for modern software applications, and especially for those in contentious and litigious technical fields such as telecommunications, social networks, and novel human interface methodologies.

More serious kinds of litigation can occur for software that controls physical devices such as medical equipment, avionics packages, weapons systems, and automotive controls where software failures can cause injury or deaths.

Summary and Results

Successful software projects can result from nothing more than avoiding the more serious mistakes that lead to disaster: 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) Make sure with 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

Abrain, Alain; Software Estimating Models; Wiley-IEEE Computer Society; 2015

Abrain, Alain; Software Metrics and Metrology; Wiley-IEEE Computer Society; 2010

Abrain, Alain; Software Maintenance Management: Evolution and Continuous Improvement; Wiley-IEEE Computer Society, 2008.

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

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

Boehm, Barry Dr.; Software Engineering Economics; Prentice Hall, Englewood Cliffs, NJ; 1981; 900 pages.

Brooks, Fred: The Mythical Man-Month, Addison-Wesley, Reading, Mass., 1974, rev. 1995.

Bundschuh, Manfred and Dekkers, Carol; The IT Metrics Compendium; Springer, 2005.

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.

Ebert, Christof; Dumke, Reinder, and Bundschuh; Manfred; Best Practices in Software Measurement; Springer; 2004.

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

Ewusi-Mensah, Kweku; Software Development Failures; MIT Press, Cambridge, MA; 2003; ISBN 0-26205072-2276 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.

Gack, Gary; Managing the Black Hole: The Executives Guide to Software Project Risk; Business Expert Publishing, Thomson, GA; 2010; ISBN10: 1-935602-01-9.

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.

Garmus, David & Herron, David; Measuring the Software Process: A Practical Guide to Functional Measurement; Prentice Hall, Englewood Cliffs, NJ; 1995.

Garmus, David; Russac Janet, and Edwards, Royce; Certified Function Point Counters Examination Guide; CRC Press; 2010.

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

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.

Harris, Michael D.S., Herron, David, and Iwanicki, Stasia; The Business Value of IT; CRC Press, Auerbach; Boca Raton, FL; 2008; ISBN 978-14200-6474-2.

Hill, Peter; Jones Capers; and Reifer, Don; The Impact of Software Size on Productivity; International Software Standards Benchmark Group (ISBSG), Melbourne, Australia, September 2013.

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; The Technical and Social History of Software Engineering, Addison Wesley 2015 (contains summaries of important software industry lawsuits such as anti-trust and patent violations).

Jones, Capers: "Studio 38 in Rhode Island – A Study of Software Risks"; 2012, published in various Rhode Island newspapers such as the Providence Journal, South County Independent, Narragansett Times, etc.

Jones, Capers and Bonsignour, Olivier; The Economics of Software Quality; Addison Wesley, Boston, MA; ISBN 10 0-13-258220-1; 2011; 587 pages.

Jones; Capers; Software Engineering Best Practices; McGraw Hill, New York, NY; ISBN 978-0-07-162161-8; 2010; 660 pages.

Jones, Capers; Applied Software Measurement; McGraw Hill, 3rd edition 2008; ISBN 978-0-07-150244-3; 662 pages.

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 Qualitiy Low Cost Software Inspections; Paradoxicon Publishingl 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.

Wiegers, 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

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 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.namcook.com This web site 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 less. It also contains a catalog of software benchmark providers which currently lists 20 organizations that provide quantitative data about software schedules, costs, quality, and risks.

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.