Software Tracking: The Last Defense Against Failure©

Capers Jones
Software Productivity Research, LLC

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

This article is based on software projects that were in litigation for breach of contract. It concentrates on four worst practices or the factors that most often lead to failure and litigation. A previous article dealt with additional problems noted during litigation [1].

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

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

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

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

Let us consider each of these topics in turn.

Estimating Problems

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

However, just because an accurate estimate can be produced using a commercial estimating tool, this 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. The other half had accurate estimates but they were rejected and replaced by forced estimates based on business needs rather than team abilities.

The main reason that accurate estimates were rejected and replaced was the absence of supporting historical data. Without this, 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.

A situation such as this was one of the contributing factors to the long delay in opening the Denver International Airport. Estimates for the length of time to complete and debug the very complex baggage handling software were not believed [2].

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[1]. Currently, more than 4,000 projects are available, and new projects are added at a rate of perhaps 500 per year.

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

Changing Requirements

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

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 percent. 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.

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

In several cases, the requirements changes had not been formally included in the contracts for development, and the clients refused to pay for changes that substantially affected the scope of the projects. One case involved 82 changes that totaled to more than 2,000 function points or about 20 percent of the original size of the initial requirements.

Since the defect potentials for changing requirements are larger than for the original requirements by about 10 percent, and since defect removal efficiency for changing requirements is lower by about 5 percent, 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.

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 it is possible. Therefore, leading companies are ready and able to deal with changes and do not let

© 2007 by Capers Jones. All rights reserved.

them become impediments to progress. For projects developed under contract, the contract itself must include unambiguous language for dealing with changes.

Quality Problems

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 it takes more time than any other activity.

Successful quality control involves defect prevention, defect removal, and defect measurement activities. The phrase *defect prevention* includes all activities that minimize the probability of creating an error or defect in the first place. Examples of defect prevention activities include the Six Sigma approach, joint application design for gathering requirements, usage of 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, and all kinds of testing.

Some activities benefit both defect prevention and defect removal simultaneously. For example, participation in design and code inspections is very effective in terms of defect removal, and also benefits defect prevention. Defect prevention is aided because inspection participants learn to avoid the kinds of errors that inspections detect.

As stated earlier, a 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 [1]. However, additional data now shows that for projects in the 10,000 function point range the successful ones accumulate development totals of around 4.0 defects per function point and remove about 95 percent of them before delivery to customers. In other words, the number of delivered defects is about 0.2 defects per function point or 2,000 total latent defects. Of these, about 10 percent or 200 would be fairly serious defects. The rest would be minor or cosmetic defects.

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

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

Software Milestone Tracking

Those readers 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 under way, 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.

For an industry now more than 50 years old, it is somewhat surprising that there is not a general or universal set of project milestones for indicating tangible progress. From the author's assessment and baseline studies, Table 1 (see next page) shows some representative milestones that have shown practical value.

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

Suggested Format for Monthly Status Reports for Software Projects

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

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.

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 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 – this kind of situation cannot be anticipated.

It might be thought that monthly reports are too far apart for small projects that last six months or less in total. For

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

Table 1: *Representative Tracking Milestones for Large Software Projects*

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 in litigation except one was larger than 10,000 function points in size.

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

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; they should reflect the dates that finished deliverables were validated by means of inspections, testing, and quality assurance review.

Summary and Results

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.◆

References

1. Jones, Capers. "Social and Technical Reasons for Software Project Failure." CrossTalk June 2006: 4-9.
2. Gibbs, T. Wayt. "Trends in Computing: Software's Chronic Crisis." Scientific American Magazine Sept. 1994: 72-81.

Note

1. This data is available in both CD and paper form <www.isbsg.org>.

About the Author

Capers Jones is currently the chairman of Capers Jones and Associates, LLC. He is also the founder and former chairman of Software Productivity Research (SPR) where he holds the title of Chief Scientist Emeritus. He is a well-known author and international public speaker, and has authored the books "Patterns of Software Systems Failure and Success," "Applied Software Measurement," "Software Quality: Analysis and Guidelines for Success," "Software Cost Estimation," and "Software Assessments, Benchmarks, and Best Practices." Jones and his colleagues from SPR have collected historical data from more than 600 corporations and more than 30 government organizations. This historical data is a key resource for judging the effectiveness of software process improvement methods.

Software Productivity Research, LLC
Phone: (877) 570-5459
Fax: (781) 273-5176
E-mail: capers.jones@spr.com, info@spr.com