Software Project Management Practices: Failure Versus Success©

Capers Jones
Software Productivity Research LLC

An analysis of approximately 250 large software projects between 1995 and 2004 shows an interesting pattern. When comparing large projects that successfully achieved their cost and schedule estimates against those that ran late, were over budget, or were cancelled without completion, six common problems were observed: poor project planning, poor cost estimating, poor measurements, poor milestone tracking, poor change control, and poor quality control. By contrast, successful software projects tended to be better than average in all six of these areas. Perhaps the most interesting aspect of these six problem areas is that all are associated with project management rather than with technical personnel. Two working hypotheses emerged: 1) poor quality control is the largest contributor to cost and schedule overruns, and 2) poor project management is the most likely cause of inadequate quality control.

This article is derived from analysis of about 250 large software projects at or above 10,000 function points in size that were examined by the author's company between 1995 and 2004. (Note that 10,000 function points are roughly equivalent to 1,250,000 statements in the C programming language.)

It is difficult during analysis to pick out successful or unsuccessful methods from projects that are more or less average. However when polar opposites are examined, some very interesting differences stand out. The phrase *polar opposites* refers to projects at opposite ends of the spectrum in terms of achieving cost, schedule, and quality targets. When projects that were late by more than 35 percent, or overran their budgets by more than 35 percent, or experienced serious quality problems after delivery are compared to projects without such issues, some interesting patterns can be seen.

Of the 250 projects analyzed, about 25 were deemed successful in that they achieved their schedule, cost, and quality objectives. About 50 had delays or overruns below 35 percent, while about 175 experienced major delays and overruns, or were terminated without completion. The projects included systems software, information systems, outsourced projects, and defense applications. This distribution of results shows that large system development is a very hazardous undertaking. Indeed, some of the failing projects were examined by the author while working as an expert witness in breach-of-contract litigation involving the failed projects.

These large applications included both systems software and information systems. Both corporations and government agencies were included. In terms of development methods, both waterfall development cycles and spiral development were included. The newer *agile* methods were not included because such methods are seldom if ever utilized on applications larger than about 1,000 function points.

Table 1 shows six major factors noted at opposite ends of the spectrum in terms of failure versus success as they were revealed in the study analysis.

The author and his colleagues were commissioned by clients to examine the software development practices, tools utilized, quality, and productivity results of various projects. Thus, this article may be biased toward the topics examined. We were not commissioned to examine other kinds of issues such as poor training, staff inexperience, or poor personnel practices. There are, of course, many other influential factors besides these six in this report. Indeed, several prior books by the author cited more than 100 factors [1, 2]. But these six key factors occur so frequently that they stand out from factors that occur only now and then. For additional studies on recent project failures other than the author's, see [3, 4, 5, 6].

Before dealing with the patterns observed on the successful and failing projects, it is desirable to discuss some of the differences between project planning and project estimating since these are the key factors associated with both success and failure.

The phrase *project management tools* has been applied to a large family of tools whose primary purpose is sophisticated scheduling for projects with hundreds or even thousands of overlapping and partially interdependent tasks. These tools are able to drop down to very detailed task levels, and can even handle the schedules of individual workers. A few examples of tools within the project management class include Artemis Views, Microsoft Project, Primavera, and Project Manager's Workbench.

However, the family of project management tools is general purpose in nature and does not include specialized software sizing and estimating capabilities as do the software cost estimating tools. Neither do these general project management tools deal with quality issues such as defect removal efficiency. Project management tools are useful, but software requires additional capabilities to be under full management control.

The software cost estimation industry and the project management tool industry originated as separate businesses with project management tools appearing in the 1960s, around 10 years before software cost estimating tools. Although the two were originally separate businesses,

Table 1: *Opposing Major Factors in Study Analysis*

Successful Projects	Failing Projects
Effective project planning	Inadequate project planning
Effective project cost estimating	Inadequate cost estimating
Effective project measurements	Inadequate measurements
Effective project milestone tracking	Inadequate milestone tracking
Effective project change management	Ineffective change control
Effective project quality control	Inadequate quality control

they are now starting to join together technically.

Examples of specialized software cost estimating tools include Before You Leap, CHECKPOINT, Constructive Cost Model (COCOMO) II, CostXpert, KnowledgePlan, Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM), and Software Life Cycle Management (SLIM).

Project management tools are an automated form of several techniques developed by the Navy for controlling large and complex weapons systems. For example, the *program evaluation and review technique* (PERT) originated in the 1950s for handling complex military projects such as building warships. Other capabilities of project management tools include critical path analysis, resource leveling, and production of Gantt or timeline charts. There are many commercial project management tools available such as Artemis Views, Microsoft Project, Primavera, Project Manager's Workbench, and more.

Project management tools did not originate for software, but rather for handling very complex scheduling situations where hundreds or even thousands of tasks need to be determined and sequenced, and where dependencies such as the completion of a task might affect the start of subsequent tasks.

Project management tools have no built-in expertise regarding software as do the commercial software cost estimating tools. For example, if you wish to explore the quality and cost impact of an object-oriented programming language such as Smalltalk, a standard project management tool is not the right choice.

By contrast, many software cost estimating tools have built-in tables of programming languages and will automatically adjust the estimate based on which language is selected for the application.

Since software cost estimating tools originated about 10 years after commercial project management tools, the developers of software cost estimating tools seldom tried to replicate project management functions such as construction of detailed PERT diagrams or critical path analysis. Instead, the cost estimation tools would export data to a project management tool. Thus, interfaces between software cost estimating tools and project management tools are now standard features in the commercial estimation market.

Let us now turn to applying project planning and project estimating tools to large software applications.

Successful and Unsuccessful Project Planning

The phrase *project planning* encompasses creating work breakdown structures, and then apportioning tasks to staff members over time. Project planning includes creation of various timelines and critical paths including Gantt charts, PERT charts, or the like.

Effective project planning for large projects in large corporations involves both planning specialists and automated planning tools. Successful planning for large software projects circa 2004 involves the following:

- Using automated planning tools such as Artemis Views or Microsoft Project.
- Developing complete work breakdown structures.
- Conducting critical path analysis of project development activities.
- Considering staff hiring and turnover during the project.
- Considering subcontractors and international teams.
- Factoring in time for requirements gathering and analysis.
- Factoring in time for handling changing requirements.
- Factoring in time for a full suite of quality control activities.
- Considering multiple releases if requirements growth is significant.

Successful projects do planning very well indeed. Delayed or cancelled projects, however, almost always have planning failures. The most common planning failures include (1) not dealing effectively with changing requirements; (2) not anticipating staff hiring and turnover during the project; (3) not allotting time for detailed requirements analysis; and (4) not allotting sufficient time for inspections, testing, and defect repairs.

Successful project planning tends to be highly automated. There are at least 50 commercial project-planning tools on the market, and successful projects all use at least one of these. Not only are the initial plans automated, but also any changes in requirements scope or external events will trigger updated plans to match the new assumptions. Such updates cannot be easily accomplished via manual methods; planning tools are a necessity for large software projects.

Successful and Unsuccessful Project Cost Estimating

Software cost estimating for large software projects is far too complex to be performed manually. This observation is supported by the presence of at least 75 commercial software cost estimating tools, including such well-known tools as COCOMO II, CostXpert, KnowledgePlan, PRICE-S, SEER-SEM, SLIM, and the like [7]. Successful projects all use at least one such tool, and usage of two or more is not uncommon. Estimates produced by trained estimating specialists are also noted on many successful large projects, but not on failing projects. Successful cost estimating for large systems involves using the following:

- Software estimating tools (COCOMO II, CostXpert, KnowledgePLAN, PRICE-S, SEER-SEM, SLIM, etc.).
- Formal sizing approaches for major deliverables based on function points.
- Comparison of estimates to historical data from similar projects.
- Availability of trained estimating specialists or project managers.
- Inclusion of new and changing requirements in the estimate.
- Inclusion of quality estimation as well as schedule and cost estimation.

By contrast, large failing projects may not utilize any of the commercial software estimating tools. However, manual estimates are never sufficient for projects in the 10,000-function point range.

Failing projects tend to understate the size of the work to be accomplished due to inadequate sizing approaches. Failing projects also omit quality estimates, which are a major omission since excessive defect levels slow down testing to a standstill. Overestimating productivity rates or assuming that productivity on a large system will be equal to productivity on small projects are other common reasons for cost and schedule overruns. The main problem with estimates for projects in the 10,000-function point size range is that they err on the side of excessive optimism.

Project planning tools and project estimating tools overlap in functionality, and are usually marketed separately. Normally, the project planning and cost estimating tools pass information back and forth. The software cost estimating tool would be used for overall project sizing, resource estimating, and quality estimating. The project-planning tool would be used for critical path analysis, detailed scheduling, and for work breakdown structures.

Successful and Unsuccessful Project Measurements

Successful large projects are most often found in companies that have software measurement programs for capturing productivity and quality historical data [8, 9]. Thus any new project can be compared against similar projects to judge the validity of schedules, costs, quality, and other important factors. The most useful measurements for projects in the 10,000-function point domain include measures of the following:

- Accumulated effort.
- Accumulated costs.
- Development productivity.
- Volume and rate of requirements changes.
- Defects by origin.
- Defect removal efficiency.

Measures of effort should be granular enough to support work breakdown structures. Cost measures should be complete and include development costs, contract costs, and costs associated with purchasing or leasing packages. There is one area of ambiguity even for top companies and successful projects: The overhead or burden rates established by companies vary widely. These variances can distort comparisons between companies, industries, and countries, and make benchmarking difficult. Of course, within a single company this is not an issue.

Function points are now the most commonly used metric in both the United States and Europe for software projects, and are rapidly growing in usage throughout the world. Development productivity measurements normally use function points in two fashions: function points per staff month and/or work hours per function point [10, 11, 12]. For additional information on functional metrics, refer to the Web site of the non-profit International Function Point Users Group at <www.ifpug.org>.

The federal government, some military projects, and the defense industry still perform measurements using the older *lines-of-code* metric. This metric is hazardous because it cannot be used for measuring many important activities such as requirements, design, documentation, project management, quality assurance, and the like. There are also programming languages such as Visual Basic that have no effective rules for counting lines of code. About one third of the large software projects examined utilized several programming languages concurrently, and one large application included 12 different programming languages.

Measures of quality are powerful indicators of top-ranked software producers and are almost universal on successful projects. Projects that are likely to fail, or have failed, almost never measure quality. Quality measures include defect volumes by origin (i.e., requirements, design, code, bad fixes) and severity level, defect severity levels, and defect repair rates.

Really sophisticated companies and projects also measure defect removal efficiency. This requires accumulating all defects found during development and also after release to customers for a predetermined time period. For example, if a project finds 900 defects during development and the users find 100 defects in the first three months of use, then it can be stated that the project achieved a 90 percent defect removal efficiency level. Of course, any defect found after the first three months lowers the defect removal value.

It is interesting that successful projects are almost always better than 95 percent in defect removal efficiency, which is about 10 percent better than the U.S. average of 85 percent [13].

It is not possible to measure defect removal efficiency for cancelled projects since there is no customer usage. However, for projects that finally get released to customers – although delivered late – defect removal efficiency seldom tops 80 percent, or about 5 percent below U.S. averages and 15 percent below successful projects. This statement is based on only about a dozen large systems because almost universally, projects that are delayed or over budget do not have effective quality measurements in place.

"... it can be hypothesized that lack of effective quality control on large systems is a major contributor to both cost and schedule overruns."

Since the bulk of schedule delays and cost overruns tends to occur during testing and is caused by excessive defect volumes, it can be hypothesized that lack of effective quality control on large systems is a major contributor to both cost and schedule overruns.

Successful and Unsuccessful Milestone Tracking

The phrase *milestone* tracking is ambiguous in the software world. It sometimes refers to the start of an activity, sometimes to the completion of an activity, and sometimes to nothing more than a calendar date. In this article, the phrase refers to the point of formal completion of key deliverables or a key activity. Normally, a completion milestone is the direct result of some kind of review or inspection of the deliverable. A milestone is not an arbitrary calendar date.

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.

A typical set of project milestones for successful software applications in the nominal 10,000-function point size range would include completion of the following:

- Requirements review.
- Project plan review.
- Cost and quality estimate review.
- External design reviews.
- Database design reviews.
- Internal design reviews.
- Quality plan and test plan reviews.
- Documentation plan review.
- Deployment plan review.
- Training plan review.
- Code inspections.
- Each development test stage.
- Customer acceptance test.

Failing or delayed projects usually lack serious milestone tracking. Activities might be reported as finished while work was still ongoing. Milestones might be simple dates on a calendar rather than completion and review of actual deliverables. Some kinds of reviews may be so skimpy as to be ineffective.

Successful projects, on the other hand, regard milestone tracking as an important activity and try to do it well. There is no glossing over of missed milestones, or pretending that unfinished work is done. Delivering documents or code segments that are incomplete, contain errors, and cannot support downstream development work is not the way milestones occur on successful projects.

Another aspect of milestone tracking on successful projects is what happens when problems are reported or delays occur. The reaction is strong and immediate: corrective actions are planned, task forces assigned, and corrections occur as rapidly as possible. Among lagging projects, on the other hand, problem reports may be ignored and very seldom do corrective actions occur.

Successful and Unsuccessful Change Management

Applications in the nominal 10,000-function point size range run from 1 percent to 3 percent per month in new or changed requirements during the analysis and design phases [8]. This fact was discovered by measuring the initial function point totals at the requirements stage and comparing them to the function point total after design. If the initial function point total is 10,000 function points and the post-design total is 12,000 function points, then the overall growth is 20 percent. If the schedule for analysis and design took 10 calendar months, then the monthly growth rate was 2 percent per month.

The total accumulated volume of changing requirements can top 50 percent of the initial requirements when function point totals at the requirements phase are compared to those at deployment. Therefore, successful software projects in the nominal 10,000-function point size range must use state-of-the-art methods and tools to ensure that changes do not get out of control.

Successful change control for applications in the 10,000-function point size range include the following:

- A joint client/development change control board or designated domain experts.
- Using joint application design (JAD) to minimize downstream changes.
- Using 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 greater than 10 function points.
- Prioritizing change requests in terms of business impact.
- Formal assignment of change requests to specific releases.
- Using automated change control tools with cross-reference capabilities.

One of the observed byproducts of using formal JAD sessions is a reduction in downstream requirements changes. Rather than having unplanned requirements surface at a rate of 1 percent to 3 percent per month, studies of JAD by IBM and other companies have indicated that unplanned requirements changes often drop below 1 percent 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.

However, 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. Therefore, some form of iterative development is a logical necessity.

Successful and Unsuccessful Quality Control

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

Successful quality control involves both defect prevention and defect removal 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 JAD for gathering requirements, using formal design methods, using structured coding techniques, and using 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. The reason why defect prevention is aided is because inspection participants learn to avoid the kinds of errors that inspections detect. Successful quality control activities for 10,000-function point projects include the following:

Defect Prevention

- JAD for gathering requirements.
- Formal design methods.
- Structured coding methods.
- Formal test plans.
- Formal test case construction.

Defect Removal

- Requirements inspections.
- Design inspections.
- Document inspections.
- Code inspections.
- Test-plan and test-case inspections.
- Defect repair inspections.
- Software quality assurance reviews.
- Unit testing.
- Component testing.
- New function testing.
- Regression testing.
- Performance testing.
- System testing.
- Acceptance testing.

The combination of defect prevention and defect removal activities leads to some very significant differences in the overall numbers of software defects compared between successful and unsuccessful projects. For projects in the 10,000-function point range, the successful ones accumulate development totals of around 4.0 defects per function point and remove about 95 percent of them before customer delivery. 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 about 20 percent or 2,800 would be fairly serious defects. This large number of latent defects after delivery is very troubling for users.

One of the reasons why successful projects have such high defect removal efficiency compared to unsuccessful projects is the usage of design and code inspections [14, 15]. Formal design and code inspections average about 65 percent efficiency in finding defects. They also improve testing efficiency by providing better source materials for constructing test cases.

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 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.

(Note: The term *bad fixes* refers to secondary defects accidentally injected by means of a patch or defect repair that is itself flawed. The industry average is about 7 percent, but for unsuccessful projects the number of bad fixes can approach 20 percent; i.e., one out of every five defect repairs introduced fresh defects [13]. Successful projects, on the other hand, can have bad-fix injection rates of only 2 percent or less.)

Conclusions

There are many ways to make large software systems fail. There are only a few ways of making them succeed. It is interesting that project management is the factor that tends to push projects along either the path to success or the path to failure.

Large software projects that are inept in quality control and skimpy in project management tasks are usually doomed to either outright failure or massive overruns.

Among the most important software development practices leading to success are those of planning and estimating before the project starts, absorbing changing requirements during the project, and successfully minimizing bugs or defects.

Successful projects always excel in these critical activities: planning, estimating, change control, and quality control. By contrast, projects that run late or fail typically had flawed or optimistic plans, had estimates that did not anticipate changes or handle change well, and failed to control quality.◆

References

1. Jones, Capers. Patterns of Software Systems Failure and Success. Boston, MA: International Thompson Computer Press, 1995.
2. Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston, MA: Addison Wesley Longman, 2000.
3. Ewusi-Mensah, Kweku. Software Development Failures. Cambridge, MA: MIT Press, 2003.
4. Glass, R.L. Software Runaways: Lessons Learned From Massive Software Project Failures. Englewood Cliffs, NJ: Prentice Hall, 1998.
5. The Standish Group. The CHAOS Chronicles Vers. 3.0. West Yarmouth, MA: The Standish Group, 2004.
6. Yourdon, Ed. Death March – The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects. Upper Saddle River, NJ: Prentice Hall PTR, 1997.
7. Jones, Capers. Estimating Software Costs. New York: McGraw Hill, 1998.
8. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality. 2nd ed. New York: McGraw Hill, 1996.
9. Kan, Stephen H. Metrics and Models in Software Quality Engineering. 2nd ed. Boston, MA: Addison Wesley Longman, 2003.
10. Garmus, David, and David Herron. Function Point Analysis – Measurement Practices for Successful Software Projects. Boston, MA: Addison Wesley Longman, 2001.
11. International Function Point Users Group. IT Measurement – Practical Advice from the Experts. Boston, MA: Addison Wesley Longman, 2002.
12. Jones, Capers. "Sizing Up Software." Scientific American Magazine 279.6 (Dec. 1998).
13. Jones, Capers. Software Quality – Analysis and Guidelines for Success. Boston, MA: International Thomson Computer Press, 1997.
14. Radice, Ronald A. High Quality Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing, 2002.
15. Wiegers, Karl E. Peer Reviews in Software – A Practical Guide. Boston, MA: Addison Wesley Longman, 2002.

About the Author

Capers Jones is founder and chief scientist of Software Productivity Research LLC. He is an international consultant on software management topics, a speaker, a seminar leader, and author. Jones was formerly at the ITT Programming Technology Center in Stratford, Conn., where he was assistant director of Programming Technology. Prior to joining ITT, he was at IBM for a 12-year period in both research and managerial positions. He received the IBM General Product Division's outstanding contribution award for his work in software quality and productivity improvement methods. Jones has published 12 books on software project management topics and more than 200 journal articles. He has given seminars on software project management in more than 20 countries to more than 150 major corporations, government agencies, and military services.

Phone: (401) 789-7662
Fax: (401) 782-2755
E-mail: cjones@spr.com

Coming Events

November 3-5
Association for Computing Machinery SenSys '04
Baltimore, MD
www.cse.ohio-state.edu/sensys04

November 7-10
Amplifying Your Effectiveness AYE 2004
Phoenix, AZ
www.ayeconference.com

November 8-13
13th Conference on Information and Knowledge Management CIKM 2004
Washington, D.C.
www.ir.iit.edu/cikm2004

November 14-18
SIGAda 2004
Atlanta, GA
www.acm.org/sigada/conf/sigada2004

November 15-18
4th Annual Capability Maturity Model® Integration Technology Conference
Denver, CO
www.sei.cmu.edu/cmmi/events/cmmi-techconf.html

November 15-19
Software Testing Analysis and Review STARWEST '04
Anaheim, CA
www.sqe.com/starwest/

November 17-19
2004 Federal CTO Summit
Washington, DC
www.federalctosummit.com

April 18-21, 2005
2005 Systems and Software Technology Conference

STC Systems & Software Technology Conference

Salt Lake City, UT
www.stc-online.org