

THE IMPACT OF POOR QUALITY AND CANCELED PROJECTS ON THE SOFTWARE LABOR SHORTAGE

Version 3.0 - July 12, 2005

Abstract

When the work patterns of software development and maintenance projects are analyzed, a surprising hypothesis emerges. There would not be a significant software personnel shortage if the current level of software errors could be reduced or eliminated. The amount of software effort spent on software projects that will be canceled due to excessive error content appears to absorb more than 15% of the global software work force. In addition, about 60% of the U.S. software work force are engaged in fixing errors, which might have been avoided.

Somewhere between about 47 and 97 days a year of the U.S. software work force are available for actual software development tasks over and above the work of fixing errors. No other major occupation appears to devote so much effort to canceled projects or defect repairs as does software.

A stronger focus on software quality control and risk analysis would seem to offer substantial economic advantages, by reducing the amount of software engineering effort currently spent on defect repairs and projects too poorly constructed to be completed.

Capers Jones, Chief Scientist Emeritus
Software Productivity Research, Inc.
Phone 401 789-7662
Email CJones@SPR.com
Web <http://www.spr.com>

Copyright © 1998 - 2005 by Capers Jones. All Rights Reserved.

INTRODUCTION

Since about 1998 the software industry has faced a significant labor shortage. Although some academics have debated the reality of this shortage, it is quite apparent to software managers and to the commercial software industry. During the period from 1998 through 2000 many software projects were delayed or deferred while software professionals worked on year 2000 repairs or on updating software in support of the Euro. These two massive projects triggered increased demand for software personnel. They also led to an increase in software salaries throughout the United States and Western Europe.

Software is one of the most labor-intensive occupations of the 20th century. Software is also one of the most challenging business endeavors, since software projects are difficult to control and subject to a significant percentage of delays and outright cancellations. The primary reason for both software delays and cancellations is due to the large numbers of "bugs" or errors whose elimination can absorb more than half of the effort on really large software projects. Defect removal is almost always the largest identifiable source of software cost and schedule overruns.

When delays in software schedules and overages in software cost estimates are analyzed carefully, it can be seen that a major source of schedule slippage and cost overruns is the fact that the applications have so many bugs that they don't work or can't be released. A prime example of this phenomenon can be seen in the one-year delay in opening the Denver Airport due to errors in the software controlling the luggage handling system. Problems with excessive bugs or errors have also caused delays in many software product releases, even by such well-known companies as IBM and Microsoft.

A secondary source of estimating error is the increase in unplanned "creeping" requirements during the development cycle. The use of function point metrics has allowed "requirements creep" to be measured directly. The current U.S. average for creeping requirements is about 2% per month after the requirements phase through the design and coding phases, based on the approximate 10,000 projects examined by Software Productivity Research (SPR) in the course of our assessment and benchmark studies (Jones 1996).

Creeping requirements added during the design and coding phases tend to generate far more than their share of errors. Thus the root cause of delays and overruns from requirements creep is in reality due to unexpected bugs or errors. These slow down testing and stretch out schedules. Thus unplanned or creeping requirements generate delays at both ends: During early development unplanned requirements stretch schedules as they occur, and during testing unplanned requirements stretch schedules as their bugs are discovered and repaired.

From analysis of the long-range patterns of software engineering work activities, an interesting hypothesis occurs. A key reason for the current software labor shortage is due to software engineering time spent either on projects that will be cancelled before completion, or time spent in the pursuit of bugs or defects which might be avoided. This hypothesis leads to another hypothesis. Better quality control and better software project risk management could reduce the labor shortage within a period of 5 to 10 years.

Analyzing the Work Patterns of Software Engineers and Programmers

Software engineering is a very labor-intensive occupation. A key reason for the high labor content of software applications is because these applications are very complex and hence very error-prone.

The large number of severe errors or “bugs” in software applications has several unfortunate effects on the software industry:

1. A substantial number of software projects are cancelled due to high error rates.
2. Much of the development work of software engineering is defect removal.
3. Much of the maintenance work of software engineering is defect repair.

In 1998 as part of a study of U.S. software economics, the author used data from SPR’s software assessments and benchmark studies to extrapolate an approximate national overview of software projects that were underway. The assumption was that the distribution of projects among SPR’s client companies could serve as a surrogate for the U.S. as a whole. Between 1998 and 2005 there has been only a small increase in on-going U.S. projects, due in part to outsourcing overseas.

Table 1 shows the approximate number of software “projects” that were underway in the United States during calendar year 1998. A software “project” was defined as the total effort assigned to developing or enhancing a specific software application.

Table 1 uses the function point metric as a way of normalizing the size of software applications. Six size plateaus are indicated, each separated by an order of magnitude. This is a simplifying assumption, and applications of other sizes are assigned to the closest size plateau.

Table 1: U.S. Software Projects in Progress in the Year 1998

Size in Function Points	Projects in Progress	Projects Cancelled	Percent Cancelled
1	2,000,000	5,000	0.25%
10	2,500,000	50,000	2.00%
100	1,500,000	109,950	7.33%
1000	100,000	20,330	20.33%
10000	2,500	1,200	48.00%
100000	50	33	65.00%
TOTAL	6,102,550	186,513	3.06%

The data on the percent of projects cancelled within each size range is taken from the author’s 1995 book on [Patterns of Software Systems Failure and Success](#) (Jones 1995). Very few small projects are canceled but many large systems are terminated without being completed. Although many reasons are associated with cancelled projects, the root cause for most is that they overran both budgets and schedules due to excessive error content.

The next topic of interest was to quantify the resources associated with these six size plateaus, in order to determine the amount of software effort that appeared to be wasted on projects that do not reach completion. Table 2 shows the approximate numbers of U.S. software personnel assigned to the projects within each of the six main size plateaus. Obviously large systems, some of which have hundred of software engineers, absorbed the bulk of the available software personnel.

Table 2: Staffing Levels of U.S. Software Projects in 1998

Size in Function Points	Staff on Projects in Progress	Staff on Cancelled Projects	Percent of Staff on Cancellations
1	13,333	33	0.25%
10	166,667	3,333	2.00%
100	1,000,000	73,300	7.33%
1000	666,667	135,533	20.33%
10000	166,667	80,000	48.00%
100000	33,333	21,667	65.00%
TOTAL	2,046,667	313,867	15.34%

Tables 1 and 2 have a high margin of error of course. But if they are even slightly close to reality they present a troubling picture. Because cancelled projects are more frequent for large applications than for small, more than 15% of the available software personnel in the United States appear to work on projects that will not be completed. This is true in 2005 as it was in 1998.

Errors or bugs drive up the costs and lengthen the schedules of all major software engineering projects. Indeed, the effort for finding and fixing bugs is greater than for any other activity for civilian software projects.

Another important topic is the effort devoted to late project changes that were not discovered during the requirements phase and hence emerged downstream during design or even coding. These later changes absorb a very large amount of effort. The measured rate of these changes is about 2% per calendar month.

Table 3 illustrates the approximate distribution of time in a 365-day calendar year for typical professional software engineers circa 1998, but still true in 2005.

Table 3: Distribution of Effort in 1998 Software Work Year

Activities	Work Days	Percent
Testing and defect repairs	70	19.18%
Time on late project changes	50	13.70%
Productive time on projects	47	12.88%
Time on canceled projects	30	8.22%
Vacations and personal time	15	4.11%
Business meetings	15	4.11%
Training and Education	10	2.74%
Public holidays	9	2.47%
Travel	6	1.64%
Slack time between tasks	5	1.37%
Sick leave	4	1.10%
Weekend overtime	4	1.10%
Regular Weekends	100	27.40%
TOTAL	365	100.00%

Although the data was first gathered in 1998, there are very few changes in 2001 so the overall picture is still comparatively stable. In a full 365 day calendar year, the time spent not at work such as weekends, vacations, and holidays need to be removed in order to understand the overall work patterns of software engineers.

Table 4 backs out weekends, vacations, holidays, sick days, training, department meetings, and all other activities other than the time spent actually working on software applications.

Table 4: 1998 Software Effort on Project-Related Tasks

Activities	Work Days	Percent
Testing and defect repairs	70	35.53%
Late project changes	50	25.38%
Time on canceled projects	30	15.23%
Productive time on projects	47	23.86%
TOTAL	197	100.00%

The implications of table 4 are quite surprising. Only about 47 working days in calendar year 1998 were available for actually developing or enhancing software applications as they were originally planned and specified. About 50 days is spent on changes to the project that were not in the original plans and specifications; i.e. requirements growth.

Although table 4 has a large margin of error, it does lead to an intriguing hypothesis. There would probably be no software labor shortage if software quality could be

brought under full control and requirements were defined fully before the project commenced.

Another way of considering the issue of the time spent on software defect repairs and cancelled projects is to consider what the impact might be if the results were expressed in a more detailed fashion by considering some of the many specialized occupations that comprise the overall software domain.

The author and his colleagues were commissioned by AT&T in 1996 to do a study of software demographics in large corporations. The overall results were published in Software Assessments, Benchmarks, and Best Practices (Jones 2000). Table 5 uses the distribution of specialists to examine the relative amounts of time available for new projects versus time spent on defect repairs and cancelled projects.

Table 5: U.S. Software Work Force on Defect Repairs and New Projects

Software Occupation Groups	Number Employed	Staff for Repairs	Staff for New Work	Percent of Total
Programmer/analyst	400,000	240,000	160,000	40.00%
Programmer (maintenance)	350,000	297,500	52,500	15.00%
Programmer (development)	275,000	151,250	123,750	45.00%
Project manager (1st level)	225,000	123,750	101,250	45.00%
Software engineer (systems)	200,000	130,000	70,000	35.00%
Testing specialist	125,000	112,500	12,500	10.00%
Systems analyst	100,000	40,000	60,000	60.00%
Software engineer (realtime)	75,000	45,000	30,000	40.00%
Software technical writer	75,000	15,000	60,000	80.00%
Software engineer (embedded)	70,000	45,500	24,500	35.00%
Data administration specialist	50,000	25,000	25,000	50.00%
Project manager (2nd level)	35,000	17,500	17,500	50.00%
Software Quality Assurance	25,000	22,500	2,500	10.00%
Configuration control specialist	15,000	9,000	6,000	40.00%
Performance specialists	7,500	5,250	2,250	30.00%
Project manager (3rd level)	5,000	2,500	2,500	50.00%
Software Architect	1,500	600	900	60.00%
TOTAL	2,034,000	1,282,850	751,150	36.93%

As can be seen, for occupations such as "testing specialist" and "quality assurance" almost all of the effort is focussed on defect repairs. Even for occupations such as "development programmer" the time spent dealing with requirements changes, software error rates, and cancelled projects absorbs more than half of the available personnel.

The overall conclusion that can be drawn from table 5 is that “wastage” and defect repairs absorb almost two-thirds of the United States software workforce, leaving only about one third for productive work on new projects. So far as can be determined the software industry may be unique as the only industry where roughly two thirds of the total employees are engaged in rework and repairs, leaving only one third for productive work.

If table 5 is close to reality, then a new hypothesis about the software labor shortage can be put forth: the software labor shortage is due in large part to problems with quality control. If poor quality is a root cause of the labor shortage, then it can be asserted that more effective software quality control methods might reduce the shortage over time.

Software Defect Potentials and Defect Removal Efficiency Levels

As a first stage in exploring methods for improving software quality levels, it is important to measure the numbers of errors or defects that occur in software projects. U.S. ranges of software defects were published in the author’s book Software Quality – Analysis and Guidelines for Success (Jones 1996). Table 6 uses that data to show the range of results between lagging, average, and leading software projects from among our client base:

Table 6: Software Defects and Defect Removal Efficiency Levels In Leading, Average, and Lagging Software Organizations

(Data Expressed in terms of Defects per Function Point)

	Leading	Average	Lagging
Requirements	0.55	1.00	1.45
Design	0.75	1.25	1.90
Coding	1.00	1.75	2.35
User Manuals	0.40	0.60	0.75
Bad Fixes	0.10	0.40	0.85
TOTAL	2.80	5.00	7.30
Removal %	95%	85%	75%
Delivered	0.14	0.75	1.83

Note that errors in software requirements and software design documents often outnumber errors in the source code itself. Not only are requirements and design errors more numerous, but they are also more severe and more difficult to remove. Front-end errors in requirements and design cannot be found and removed via testing, but instead need pre-test reviews and inspections.

The basic message of table 5 is that average and lagging enterprises have very high levels of software errors or defects, and remove less than 85% of them prior to deployment. The combination of high levels of potential defects and low levels of defect removal efficiency contributes to both cancelled projects and to the dominance of error-related work patterns among the software community.

During development testing and defect removal are the most time consuming and expensive activities. After deployment, defect repairs in delivered applications is also a major consumer of software engineering effort. Here too, the conclusion is that poor software quality is a primary contributing factor to the software personnel shortage.

In other words, the software industry is locked into a pattern where more effort is associated with defect removal than with actual product development. Let us now consider some of the approaches that might be used to improve software quality and free more software resources for productive work.

IMPROVING SOFTWARE QUALITY AND CHANGE CONTROL

There are major variances from company to company and country to country in the sets of tools and methodologies used to approach software quality deal with change control. However, the best in class organizations have a common nucleus which includes these factors:

Risk Analysis

Large software applications last a very long time. Therefore each major software application should include a formal risk analysis report as part of the initial requirements. The risk analysis should include information on projected life expectancy; on connections to other applications; and on whether the application poses any potential risks to the enterprise that will use it.

Change Control Boards

Since all major projects tend to grow it is a "best practice" to establish a change control board for all projects larger 10,000 function points in size.

Joint Application Design (JAD)

Since most late requirements were missed during the requirements phase, formal methods such as joint application design can slow the rate of unplanned changes from more than 2% to a fraction of 1% per month.

Quality Measurements

The most striking difference between leading organizations and lagging ones in every country is that, without exception, the leaders know their quality levels and user satisfaction levels because they measure these factors very carefully.

The quality measurements in leading companies vary slightly, but usually include these elements: 1) Software defect volumes are measured from requirements or design throughout the rest of the development cycle and into the field; 2) Defect severity levels are measured, ranging from serious through minor; 3) Defect origins are measured, so that problems with requirements, design, code, documents, and secondary problems are known.

This software quality data is collected on a daily basis, and then summarized at monthly, quarterly, and annual intervals to show trends over time. In addition, the

leaders also measure user satisfaction, although the frequency of user surveys is normally once or twice a year.

Quality Methods

The leading companies did not become good overnight. Most of them have been engaged in software quality control work for 20 years or more. Therefore the leading companies have developed a set of proven methods that are known to work. These methods are sometimes defined under two headings, *defect prevention* and *defect removal*. Here are some examples: 1) Formal inspections of design, code, and other deliverables are used by essentially all software quality leaders since these activities are highly effective in both preventing and removing software defects; 2) Active and energetic software quality assurance groups, which may exceed 5% of total staff, are often found in the industry leaders.

Both industry leaders and laggards test their software. The most striking difference between leaders and laggards is what the leaders do before testing begins. By means of defect prevention approaches such as Joint Application Design (JAD), Quality Function Deployment (QFD), formal inspections, and various flavors of structured analysis and design, the leaders usually have far fewer problems attributable to the front of their software development life cycles. Therefore when testing begins, the code developed by the leaders is substantially free from serious problems long before testing even starts. This translates into quicker testing cycles and fewer delays of final delivery.

Ascending The Software Engineering Institute Capability Maturity Level

One of the key features of the well-known “capability maturity model” (CMM) developed by the Software Engineering Institute (SEI) is emphasis on high quality as maturity improves.

Our studies of several hundred projects at SEI levels 1, 2, 3, 4, and 5 do indicate a general improvement in defect removal efficiency at higher SEI CMM levels. Most projects by Level 1 organizations are below 75% in cumulative defect removal efficiency. Most projects by level 3 organizations or higher are above 95% in cumulative defect removal efficiency. Our recent findings on the CMM’s impact on quality were published in Software Assessments, Benchmarks, and Best Practices (Jones 2000). Table 7 illustrates the author’s findings of software quality by CMM level for projects of a nominal 5,000 function points in size:

**Table 7: Software Quality and the SEI Capability Maturity Model (CMM)
For Projects of 5000 Function Points in Size**

CMM Level	Defect Potential per Function Point	Defect Removal Efficiency	Delivered Defects per Function Point
SEI CMM 1	5.50	73.00%	1.49
SEI CMM 2	4.00	90.00%	0.40
SEI CMM 3	3.00	95.00%	0.15
SEI CMM 4	2.50	97.00%	0.08
SEI CMM 5	2.25	98.00%	0.05

One of the reasons why projects developed by the higher CMM levels have better schedule adherence and cost control is because they have better quality levels. Project failures are also reduced for the higher CMM levels.

However the SEI maturity level concept is not a panacea in terms of quality. There is some overlap among the various SEI levels. Indeed, the worst-quality software that is created by CMM level 3 organizations in can lag the best-quality software created by level 1 organizations.

Quality Assurance and Testing Tools

What is easily the most visible difference between industry quality leaders and quality laggards is the set of tools available to the leaders, and totally absent from the lagging organizations. The leaders usually employ a set of quality tools that include some or all of the following: 1) Quality estimation predictive tools; 2) Defect and quality measurement tools; 3) Test planning tools; 4) Test coverage analysis tools; 5) Software reliability predictive models; 6) Complexity analysis tools; 7) Statistical analysis and reporting tools.

These tools have the general characteristic of putting quality in tangible, quantitative terms so that the underlying root causes can be explored and improved. The laggards tend to have no quantitative data, and hence are unable to take any kind of carefully planned corrective actions.

When the software quality assurance tool suites are examined, one of the most striking differences of all springs into focus. Essentially the projects and companies in the "laggard" set have no software quality assurance function at all, and hence no SQA tool suites either.

By contrast, the leaders in terms of delivery, schedule control, and quality all have well-formed independent software quality assurance groups that are supported by powerful and growing tool suites. Unfortunately, even leading companies are sometimes understaffed and have too few quality personnel for the work at hand. In part, this is due to the fact that so few companies have software measurement and

metrics programs in place that the significant business value of achieving high levels of software quality is often unknown to the management and executive community.

Table 8 illustrates the sets of quality assurance tools noted in lagging, average, and leading organizations in terms of both numbers of tools and their approximate function point totals:

Table 8: Numbers and Size Ranges of Software Quality Tools
(Size data expressed in terms of function point metrics)

Quality Assurance	Lagging	Average	Leading
Quality estimation			2,000
Data quality analysis			1,250
QFD support			1,000
TQM support			1,000
Inspection support			1,000
Reliability estimation			1,000
Defect tracking		750	1,000
Complexity analysis		500	1,000
<i>Function point subtotal</i>	<i>0</i>	<i>1,250</i>	<i>9,250</i>
<i>Number of tools</i>	<i>0</i>	<i>2</i>	<i>8</i>

Several tools in the quality category are identified only by their initials, and need to have their purpose explained. The set of tools identified as “QFD support” are those which support the special graphics and data analytic methods of the “quality function deployment” methodology.

The set of tools identified as “TQM support” are those which support the reporting and data collection criteria of the “total quality management” methodology.

The other tools associated with the leaders are the tools of the trade of the software quality community: tools for tracking defects, tools to support design and code inspections, quality estimation tools, reliability modeling tools, and complexity analysis tools.

Complexity analysis tools are fairly common, but their usage is much more frequent among the set of leading projects than among either average or lagging projects.

Unfortunately, since the laggards tend to have no quality assurance tools at all, the use of ratios is not valid in this situation. In one sense, it can be said that the leading projects have infinitely more software quality tools than the laggards, but this is simply because the lagging set often have zero quality tools deployed.

In most companies the testing organization and the quality assurance organization are under different management, and often use different tool suites. Although there are significant differences between the leading and lagging projects in terms of testing tools, even the laggards test their software and hence have some testing tools available.

Note that there is some overlap in the tools used for testing and the tools used for quality assurance. For example, both test teams and software quality assurance teams may both utilize complexity analysis tools. Table 9 illustrates testing tool usage:

Table 9: Numbers and Size Ranges of Software Testing Tools
(Size data expressed in terms of function point metrics)

Testing	Lagging	Average	Leading
Test case generation			1,500
Complexity analysis		500	1,500
Year 2000 test support		500	1,500
Data quality analysis			1,250
Defect tracking	500	750	1,000
Test library control	250	750	1,000
Performance monitors		750	1,000
Capture/playback		500	750
Test path coverage	100	200	350
Test case execution		200	350
<i>Function point subtotal</i>	<i>850</i>	<i>4,150</i>	<i>10,200</i>
<i>Number of tools</i>	<i>3</i>	<i>8</i>	<i>10</i>

The differences in numbers of test tools deployed ranges by about 3 to 1 between the leading and lagging projects. However, the tool capacities vary even more widely, and the range of tool volumes is roughly 12 to 1 between the leaders and the laggards.

This is one of the more interesting differentials because all software projects are tested and yet there are still major variations in numbers of test tools used and test tool capacities. The leaders tend to employ full-time and well-equipped testing specialists while the laggards tend to assign testing to development personnel, who are often poorly trained and poorly equipped for this important activity.

For a more extensive discussion of the differences between leaders and laggards in terms of both quality assurance and testing refer to the author's book Software Quality - Analysis and Guidelines for Success (Jones 1996).

To summarize these findings, there are very significant differences in the quality tool suites deployed between leading enterprises and lagging enterprises. The data presented here is derived from the assessment and benchmark studies that SPR performs, which examines roughly 100 software projects each month.

Quality Culture

A final aspect which separates the laggards from the leaders is the culture of quality among the leaders, and its absence among the laggards. The word "culture" does not have a very precise definition, so in this context the meaning is the following: when visiting the industry leaders, almost everyone you talk to cares about quality and most of them also know something about it.

When visiting the laggards, some people care about quality and a few people know how it might be achieved, but these quality-conscious people often feel isolated and even angry that their executives have no particular interest in the subject.

There is no substitute for executive awareness of the importance of quality. When you meet an executive vice president or a CEO that can carry on a serious conversation about software quality, you can be fairly sure that the company is a pretty good one. When you visit a company where the executives know nothing of quality and give the appearance of not caring either, you can be fairly sure that the company will have some tough times ahead.

SUMMARY AND CONCLUSIONS

Analysis of the work patterns of the software engineering world reveals a surprising fact. Much of the work of software engineering is basically "wasted" because it concerns either working on projects that will not be completed, or working on repairing defects that should not be present at all.

If the software community can be alerted to the fact that much of the current software labor shortage is due to poor quality, then we might be motivated to take defect prevention, defect removal, and risk analysis more seriously than they are taken today.

REFERENCES AND READINGS

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; Applied Software Measurement; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 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; ISBN 0-07-9130941; 1998; 725 pages.

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.

Paulk, Mark; Charles V., Curtis, Bill; and Chrissis, Mary Beth; The Capability Maturity Model – Guidelines for Improving the Software Process by, Addison Wesley, Reading, MA; 1995).

Rubin, Howard; Software Benchmark Studies For 1999; Howard Rubin Associates, Pound Ridge, NY; 1999.

Rubin, Howard et al; Software Productivity and Quality Issues and the United States IT Workforce Shortage; Howard Rubin Associates, Pound Ridge, NY; 1997.