Software Quality and Software Economics

THE MAIN PROBLEM WITH TROUBLED SOFTWARE PROJECTS IS THERE ARE TOO MANY SERIOUS DEFECTS. ACHIEVING STATE-OF-THE-ART QUALITY CONTROL CAN GREATLY REDUCE DEFECTS AND RESULT IN SUCCESSES.

by Capers Jones

Large software projects are hazardous business undertakings. More than half of software projects larger than 10,000 function points (about 1,000,000 lines of code) are either cancelled or run late by more than a year. When examining troubled software projects, it always happens that the main reason for delay or termination is due to excessive volumes of serious defects. Conversely, large software projects that are successful are always characterized by excellence in both defect prevention and defect removal. It can be concluded that achieving state of the art levels of software quality control is the most important single objective of software process improvements.

Developing large software systems has long been recognized as one of the most hazardous business activities of the computer era. Many large software systems are either cancelled, or experience schedule delays in excess of one calendar year and cost overruns that can exceed 100%.

The author and his colleagues at Software Productivity Research have examined the results of about 13,000 software projects between 1983 and 2009. **Table 1** shows the percentages of projects that are on time, early, late, or cancelled for six size plateaus each an order of magnitude apart:

Table 1 uses function point metrics for expressing software size. (For additional information on function point metrics, it is useful to visit the web site of the non-profit International Function Point Users Group (IFPUG)). The web address is http://www.ifpug.org.)

PROBABILITY OF SELECTED OUTCOMES					
	Early	On-Time	Delayed	Canceled	Sum
1 FP	14.68%	83.16%	1.92%	0.25%	100.00%
10 FP	11.08%	81.25%	5.67%	2.00%	100.00%
100 FP	6.06%	74.77%	11.83%	7.33%	100.00%
1000 FP	1.24%	60.76%	17.67%	20.33%	100.00%
10000 FP	0.14%	28.03%	23.83%	48.00%	100.00%
100000 FP	0.00%	13.67%	21.33%	65.00%	100.00%
Average	5.53%	56.94%	13.71%	23.82%	100.00%

Table 1: Software Project Outcomes By Size of Project

As can easily be seen from **Table 1** small software projects are usually successful, but large systems are not. Why not? The main reason for the failure of large software projects is poor quality. The phrase "poor quality" in this context has two meanings:

1. Excessive numbers of defects or bugs (> 6.0 per function point)
2. Inadequate defect removal activities (< 85% defect removal efficiency)

In order to know what volume of defects might be "excessive" and what level of defect removal is "inadequate" it is useful to know current U.S. averages. **Table 2** shows defects originating in requirements, designs, code, user documents, and "bad fixes" or secondary defects. **Table 2** shows the average volumes of defects found on software projects, and the percentage of defects removed prior to delivery to customers:

Defect Origins	Defect Potentials	Removal Efficiency	Delivered Defects
Requirements	1.00	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Document	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
Total	*5.00*	*85%*	*0.75*

Table 2: Defect Removal Efficiency By Origin of Defects Circa 2009 (Data Expressed in Terms of Defects per Function Point)

Table 2 illustrates two unfortunate aspects of average software projects:

1) Large volumes of defects likely to occur; 2) Defect removal efficiency is not very good.

However, when examining the results of software projects developed by leading companies that have implemented successful process improvement programs, it can be seen that total defect volumes are lower than average, while defect removal efficiency levels are better than average. **Table 3** illustrates typical results for defect potentials and defect removal levels based on the Capability Maturity Model Integration (CMMI) developed by the Software Engineering Institute (SEI):

	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

Table 3: Software Quality and the SEI Capability Maturity Model Integration (CMMI)

As can be seen, levels 3, 4, and 5 are significantly better than U.S. averages in terms of both overall volumes of defects and defect removal efficiency levels. One of the main benefits of achieving the higher CMMI levels is better quality control, which pays off in more predictable project outcomes. This raises interesting questions as to exactly what kinds of process improvements benefit defect volumes and defect removal efficiency.

There are methods of quality improvement outside of the capability maturity model. These include the Rational Unified Process (RUP), Watts Humphrey's Team Software Process (TSP), and several forms of Agile development such as extreme programming (XP) and Crystal development.

Reducing Defect Volumes

Since the number of defects found in requirements and designs outnumber coding defects, leading companies and leading projects are very thorough in gathering requirements and in producing specifications. Of course reducing coding defects and "bad fixes" are important too.

Some of the methods noted that reduce requirements and design defects include:

- A joint client/development change control board or designated domain experts
- Use of Quality Function Deployment (QFD)
- Use of Six-Sigma for Software and/or Lean Six-Sigma
- Use of formal requirements and design inspections
- Use of joint application design (JAD) to minimize downstream changes
- Use of formal prototypes to minimize downstream 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

Note that formal inspections are effective in two distinct ways. Obviously they are effective in terms of defect removal efficiency. However participants in formal inspections spontaneously avoid making the same kinds of mistakes that inspections find. As a result, formal inspections are among the most effective methods of defect prevention.

Without use of inspections U.S. quality averages approximate 5.0 bugs per function point and about 85% in terms of defect removal efficiency. When formal inspections are added to the cycle, defect potentials gradually drop below 3.0 per function point while defect removal efficiency levels routinely top 95% and may hit 99%. This combination yields shorter development schedules and lower development costs than testing alone.

Automated static analysis is a fairly new form of defect removal that also has benefits in terms of both defect prevention and defect removal. The caveat with static analysis is that there are more than 2,500 programming languages in use circa 2009. Static analysis tools only work for perhaps 25 of the most common languages such as C, C+, Java, COBOL and a small number of others.

One interesting aspect of controlling requirements is a reduction in unplanned changes or "requirements creep." Ordinary U.S. projects average about 2% per month in new and changing requirements. Leading projects where the requirements are carefully gathered and analyzed average only a fraction of 1% per month in unplanned changes. Joint application design (JAD), prototypes, and requirements inspections are all effective in reducing unplanned requirements creep.

It happens that creeping requirements tend to be buggier than original requirements. Testing defect removal efficiency is also lower against creeping requirements. Therefore both static analysis and formal inspections are key process tools to minimize the damages that often occur from poor quality control of creeping requirements.

Raising Defect Removal Efficiency Levels

Most forms of testing are less than 35% efficient in finding

bugs or defects. However, formal design and code inspections are more than 65% efficient in finding bugs or defects and sometimes top 85%. Static analysis is also high in efficiency against many kinds of coding defects. Therefore all leading projects in leading companies utilize both formal inspections, static analysis, and formal testing. This combination is the only known way of achieving cumulative defect removal levels higher than 95%. **Table 4** illustrates the measured ranges of defect removal efficiency levels for a variety of reviews, inspections, static analysis, and several kinds of test stages:

The low defect removal efficiency levels of most forms of testing explain why the best projects do not rely upon testing alone. The best projects utilize formal design and code inspections first, static analysis, and then a multi-stage testing sequence afterwards. This combination of inspections followed by static analysis and testing leads to the shortest overall development schedules, and lowers the probabilities of project failures.

Defect Removal Activity	Ranges of Defect Removal Efficiency
Formal requirement inspections	50% to 90%
Formal design inspections	45% to 85%
Formal code inspections	45% to 85%
Static analysis (automated)	55% to 90%
Unit test (manual)	15% to 50%
Unit test (automated)	20% to 60%
New function test	20% to 35%
Regression test	15% to 30%
Integration test	25% to 40%
Performance test	20% to 40%
System test	25% to 55%
Acceptance test (1 client)	25% to 35%
Low-volume Beta test (< 10 clients)	25% to 40%
Overall cumulative ranges	70% to 99%

Table 4: Software Defect Removal Efficiency Ranges

Measuring the Economic Value of Software Quality

For more than 50 years the economic value of software quality has been poorly understood due to inadequate metrics and measurement practices. The two most common software metrics in the early days of software were "lines of code" and "cost per defect." Unfortunately both of these have serious economic flaws.

The "lines of code" metric cannot be used to measure either requirements or design defects, which collectively outnumber coding defects. It is not possible to understand the real economic value of quality if more than 50% of all defects are not included in the measurements. A more subtle problem with lines of code is that this metric penalizes high-level languages such as Java and Ruby and makes older low-level languages such as C and assembly language look better than they really are. Refer to the author's book **Applied Software Measurement** for more details of this problem.

The "cost per defect" metric actually penalizes quality and tends to achieve the lowest result for the buggiest applications. This phenomenon is due to fixed costs associated with defect removal, such as the cost of writing test cases and the cost of executing test cases. Even in situations where the application has zero defects there will still be costs for writing and executing test cases. Therefore "cost per defect" goes down as numbers of bugs go up. Refer to the author's book **Software Engineering Best Practices** for more details of this problem.

The most effective method for measuring the economic value of quality is to analyze the total cost of ownership (TCO) for software applications. It will be discovered that applications with less than about 3.0 defects per function point and > 95% in defect removal efficiency will cost about 20% less to develop than identical projects with poor quality. Their schedules will be shorter by about 15%. Annual maintenance costs will be less by about 40%. The cumulative TCO of high-quality applications from the start of the first release through five years of maintenance and enhancement will be about 30% lower than identical projects with poor quality.

One final value point is very important. For large applications > 5,000 function points in size, high quality levels will minimize the odds of failure. For poor quality, failure rates in excess of 30% can occur at 5,000 function points. For high quality projects, failure rates are usually less than 5% and cancellations are due to business reasons rather than excessive cost and schedule overruns. The economic value of excellent quality is directly proportional to application size. The larger the software application the more valuable quality becomes.

As of 2009 the overall cost drivers for software indicate why software has a bad reputation among CEO's and corporate executives. Our two top cost drivers are finding and fixing bugs and cancelled projects! It is no wonder that software is poorly regarded by corporate executives.

Table 5: The Top 15 U.S. Software Cost Drivers in Rank Order Circa 2009

1. The cost of finding and fixing bugs
2. The cost of cancelled projects
3. The cost of producing paper documents and English words

4. The cost of recovery from security flaws and attacks
5. The cost of requirements changes during development
6. The cost of programming or coding
7. The cost of customer support
8. The cost of meetings and communication
9. The cost of project management
10. The cost of application renovation
11. The cost of innovation and new kinds of features
12. The cost of litigation for cancelled projects
13. The cost of training and learning software applications
14. The cost of avoiding security flaws
15. The cost of acquiring reusable components

Table 5 is a professional embarrassment. No true engineering discipline should have defect repairs and cancelled projects as the two top cost drivers. For software engineering to become a true engineering discipline, quality control will have to be much better than it is in 2009.

Table 6 shows a hypothetical rearrangement of cost drivers that should be a goal for software engineers over the next 10 years. Our top cost driver should be innovation and designing new feature: not bug repairs. **Table 6** illustrates how costs should be apportioned circa 2019:

Table 6: The Top 15 U.S. Software Cost Drivers in Rank Order Circa 2019

1. The cost of innovation and new kinds of features
2. The cost of acquiring reusable components
3. The cost of requirements changes during development
4. The cost of programming or coding
5. The cost of training and learning software applications
6. The cost of avoiding security flaws
7. The cost of producing paper documents and English words
8. The cost of customer support
9. The cost of meetings and communication
10. The cost of project management
11. The cost of application renovation
12. The cost of litigation for cancelled projects
13. The cost of finding and fixing bugs
14. The cost of recovery from security flaws and attacks
15. The cost of cancelled projects

If software quality is improved, it should be possible to spend a much higher percentage of available funds on innovation, new features, and certified reusable materials. Today's top cost drivers of defect repairs and cancelled projects should be at the bottom of the list of cost drivers and not at the top as they are in 2009.

Summary and Conclusions

The phrase "software process improvement" is somewhat ambiguous. The phrase by itself does not indicate what needs to be improved. However from analysis of large numbers of projects that were either failures or quite successful, it is obvious that quality control is the top-ranked issue that needs to be improved. With state of the art quality control, successful projects become the norm. With inadequate defect prevention and defect removal, cancelled projects and disasters are the norm.

An occupation where failures and disasters are the top cost drivers is not a true engineering discipline. In order to become a true engineering discipline, software engineering needs better quality control, better quality measures, and better economic analysis than current norms.

References and Readings

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

Galorath, Dan; Software Sizing, Estimating, and Risk Management: When Performance is Measured Performance Improves; Auerbach Publishing, Philadelphia; 2006; ISBN 10: 0849335930; 576 pages.

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.

Gilb, Tom and Graham, Dorothy; Software Inspections; Addison Wesley, Reading, MA; 1993; ISBN 10: 0201631814.

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

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; *Software Engineering Best Practices;* McGraw Hill, 2009 (due out in November of 2009)

Jones, Capers; *Applied Software Measurement;* McGraw Hill, 3rd edition 2008; ISBN 978-0-07-150244-3; 668 pages; 3rd edition (March 2008).

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; *Software Quality – Analysis and Guidelines for Success;* International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 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.

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.

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.

About the Author

Capers Jones is currently the President and CEO of Capers Jones & Associates LLC. He is also the founder and former chairman of Software Productivity Research LLC (SPR). He holds the title of Chief Scientist Emeritus at SPR. Capers Jones founded SPR in 1984. Before founding SPR Capers was Assistant Director of Programming Technology for the ITT Corporation at the Programming Technology Center in Stratford, Connecticut. He was also a manager and researcher at IBM in California.

Capers Jones is a well-known author and international public speaker. Some of his books have been translated into six languages. All of his books are translated into Japanese and his newest books are available in Chinese editions as well. He has been the keynote speaker at the annual Japanese Symposium on Software Testing in Tokyo, the International Function Point Users Group (IFPUG), the World Congress of Quality, and the opening of the Singapore chapter of the Project Management Institute. Mr. Jones also speaks at internal corporate events for companies such as IBM, Satyam, Hewlett Packard, many others.

Capers Jones' research studies include quality estimating, quality measurement, software cost and schedule estimation, software metrics, and risk analysis. He has consulted at more than 150 large corporations and also at a number of government organizations such as NASA, the U.S. Air Force, U.S. Navy, Internal Revenue Service, and the U.S. Courts. He has also worked with several State governments.

Author Contact Information

Email: **Caper Jones:** CJonesiii@cs.com